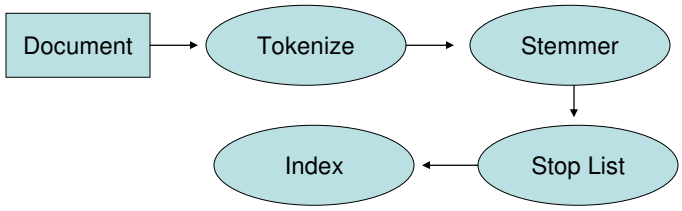
Introduction to Information Retrieval and Anatomy of Google

Information Retrieval Introduction

- Earlier we discussed methods for string matching
  - Appropriate for “small” documents that fit in memory available
  - Not appropriate for massive databases like the WWW
- The field of IR is concerned with the efficient search and retrieval of documents, often involving text
  - Documents are indexed
  - Index is searched based upon a query

Indexing

- What should be indexed?
  - Words
  - Phrases (e.g., president of the U.S.)
  - Whole sentences
- Recognizing phrases is non-trivial and does not appear to considerably improve retrieval
- Using words for indexing is the most common approach



What's a Word?

- Recognizing word boundaries is almost trivial in English
  - separated by white spaces or punctuation
- More difficult in languages such as Dutch or German, compounds are not separated by spaces
  - Pretty difficult for Arab and Asian languages
- Problems in English are mainly due to punctuation

Tokenization

- A tokenizer recognizes word boundaries
  - In . . . **sold the company.** the period is not part of company
  - . . . but in . . . in the **U.S.A.** it is
- Remove hyphens in adjective phrases such as **well-performing** system
- Some compounds are not separated by white spaces, e.g., spacewalk, waterway, household, shortcoming

Morphological Normalization

- How to make sure that **company** and **companies** and **sell** and **sold** match?
- Could ignore, but matching morphological variants increases recall
- To remove morphological information such as tense and number:
  - Stemming
    - A set of rules is used to remove suffixes
    - E.g., If word ends in ies, but not eies, aies then change ies→y
      - policies to policy
    - Cheap, but unrelated words such as **police** and **policy** can be reduced to the same stem : **polic**
  - Lemmatization
    - Use dictionary-type lookup for irregular words, more expensive
- Most search engines don't use stemming, more interested in precision (getting relevant documents you want) than recall (not missing documents that are relevant)

Stop Words

- Our document is now a list of tokenized, stemmed words or **terms**
- Some of these terms don't have much content
  - E.g. The, he, she, why, and, of
  - These words are often ignored and are called stop words
  - They can be easily filtered by using a list of commonly compiled stop words, e.g. 400 or so words
- Why eliminate stop words?
  - Using stop words does not improve retrieval
  - But reduces the size of the index considerably
  - Also can reduce retrieval time

Vector Space Document Model

- Given a collection of documents, processed into a list (i.e. vector) of terms and a query Q, also turned into a vector
  - Bag of words model: assumes each word is independent of the other
- Terms and Weights
  - Associated with each document term is a weight.
  - The weight may be binary (typically whether or not the term exists in the document), or it might be a floating point value between 0 and 1 where 0 indicates that this term is not relevant to the document, and 1 indicates that it is highly relevant to the document.

Vector Space Example

Terms:	Doc 1:	Doc 2:	Doc 3:	Query
artificial	1	1	0.3	0.5
intelligence	1	1	0.6	0.9
information	0	0	0	0.01
retrieval	1	0	0.01	0.91
mock	1	1	0.99	0.99
kenrick	0	1	0.85	0.01

Comparing Query to Documents

- Treat the Query and Document as vectors, and use standard vector comparison metrics
- Given vectors
  - $X=(x_1,x_2,\ldots, x_t)$ Document
  - $Y=(y_1, y_2,\ldots,y_t)$ Query
- Where:
  - x_i - weight of term i in the document and
  - y_i - weight of term i in the query
- For binary weights, let:
  - $|X|$ = number of 1s in the document and
  - $|Y|$ = number of 1s in query

Comparison Metrics

	For binary term vectors	For weighted term vectors
Inner product	$\lvert X \cap Y \rvert$	$\sum_{i=1}^{t} x_i y_i$
Dice coefficient	$\frac{2 \lvert X \cap Y \rvert}{\lvert X \rvert + \lvert Y \rvert}$	$\frac{2 \sum_{i=1}^{t} x_i y_i}{\sum_{i=1}^{t} x_i^2 + \sum_{i=1}^{t} y_i^2}$

Comparison Metrics

	For binary term vectors	For weighted term vectors
Cosine coefficient	$\frac{\lvert X \cap Y \rvert}{\lvert X \rvert^{1/2} \lvert Y \rvert^{1/2}}$	$\frac{\sum_{i=1}^{t} x_i y_i}{\sqrt{\sum_{i=1}^{t} y_i^2 \sum_{i=1}^{t} x_i^2}}$
Jaccard cofficient	$\frac{\lvert X \cap Y \rvert}{\lvert X \rvert + \lvert Y \rvert - \lvert X \cap Y \rvert}$	$\frac{\sum_{i=1}^{t} x_i y_i}{\sum_{i=1}^{t} x_i^2 + \sum_{i=1}^{t} y_i^2 - \sum_{i=1}^{t} x_i y_i}$

Example – Binary Inner Product

t=5000

term	1	2	…	12	…	456	…	678	…	5000
Doc-1	0	1		0		1		0		0
Doc-2	1	1		1		0		1		1
...										
Doc-N	0	1		0		1		1		0
Query	1	1		0		0		1		0

Applying the inner product of the query to doc1, doc2, and doc-N gives us
1 for doc1, 2 for doc N, and 3 for doc 2.

The ranked list is then document 2, N, and 1 with document 2 being the most relevant.

Example – Weighted Inner Product

Term	1	2	…	12	…456	…	678	…	5000
Doc-1	0	0.3		0	0.5		0		0
Doc-2	0.2	0.6		0.3	0		0.8		0.3
...									
Doc-N	0	0.2		0	0		0.6		0
Query	0.3	0.7		0	0		0.7		0

The relevance values in this case are calculated via:
For Doc-1 : 0.3*0 + 0.7*0.3 + 0.7*0 = 0.21
For Doc-2 : 0.3*0.2 + 0.7*0.6 + 0.7*0.8 = 1.04
For Doc-N : 0.3*0 + 0.7*0.2 + 0.7*0.6 = 0.56

Where do we get the weights?

- TF-IDF is a popular metric
  - Term Frequency-Inverse Document Frequency
  - Idea is to weight each term by
    - A measure of recurrence
    - A measure of term discrimination
    - A normalization factor
  - Assumes that the frequency of occurrence of a term or keyword within a document is an indicator of how relevant that term is. However, if a term or keyword appears in many documents, then its predictive power is weak.
    - E.g. word "the" occurs a lot in one document, but it's also in every single document, so it has a weak weighting

TF-IDF

- For each term, multiply the term-frequency (*tf*) by 1/document-frequency (*idf* or inverse document frequency) to obtain a metric of relevancy for each term.

$$weight(t_i) = tf(t_i) \times \log\left(\frac{N}{df(t)}\right)$$

- Where: t_i = term t in document i
- $tf(t_i)$ = Number of occurrences of term t in document i
- df(t) = Number of documents containing term t
- N = Total number of documents

TF-IDF Example

Doc 1:	Doc 2:	Doc 3:
artificial 2	artificial 3	artificial 0
intelligence 0	intelligence 5	intelligence 5
information 12	information 5	information 0
retrieval 10	retrieval 0	retrieval 0
mock 1	mock 3	mock 0
kenrick 2	kenrick 1	kenrick 0
the 35	the 56	the 42

From these frequencies we can construct tf-idf values for the terms in document 1:

Global Document Freq:		**Document 1 Freq:**		**TF-IDF Weights** $= tf(t_i) \times \log\left(\frac{N}{df(t)}\right)$
artificial	2	artificial	2	2*log(3/2)=0.35
intelligence	2	intelligence	0	0*log(3/2)=0
information	2	information	12	12*log(3/2)=2.11
retrieval	1	retrieval	10	10*log(3/1)=4.77
mock	2	mock	1	1*log(3/2)=0.17
kenrick	2	kenrick	2	2*log(3/2)=0.53
the	3	the	35	35*log(3/3)=0

Our IR System

- Vectorize all documents to tf-idf values
- Vectorize the query to tf-idf values
- Compare query to all documents using one of the comparison metrics
- Sort results by relevancy

- BUT
  - Comparing query to all documents is slow with large document sets
  - Queries on the web are on average 2.3 words
  - Use an inverted index for greater efficiency, an index on each word that tells us what documents contain that word

Inverted Indexing Example

- Objective
  - Create a sorted list of words with pointers indicating which and where the words appear in the documents.
  - We can then process the list in many different ways to meet the retrieval needs

Example

- Document D1
  - Title: Cats and dogs: Mortal enemies or simply misunderstood?
  - Category: Cats; Dogs; Children Literature
- Document D2
  - Title : New methods of feeding cats and dogs
  - Category : cats; dogs; Feeding behaviors;
- Document D3
  - Title : I Love Hot Dogs
  - Category : Children Literature; Stories; Food;

Step 1: Number all the words in each field (exclude stop words)

- **Document D1**

  T: Cats and dogs: Mortal enemies or simply misunderstood?
  1 2 3 4 5 6

  C: Cats; Dogs; Children Literature
  1 2 3 4

- **Document D2**

  T : New methods of feeding cats and dogs
  1 2 3 4 5

  C : Cats; Dogs; Feeding behaviors;
  1 2 3 4

- **Document D3**

  T : I love hot dogs
  1 2 3

  C : Children Literature ; Stories; Food;
  1 2 3 4

Step 2: make a list of words with its pointers to its document number, its field, and its position

cats	D1T01
dogs	D1T02
mortal	D1T03
enemies	D1T04
simply	D1T05
misunderstood	D1T06
cats	D1C01
dogs	D1C02
children	D1C03
literature	D1C04

new	D2T01
methods	D2T02
feeding	D2T03
cats	D2T04
dogs	D2T05
cats	D2C01
dogs	D2C02
feeding	D2C03
behaviors	D2C04

love	D3T01
hot	D3T02
dogs	D3T03
children	D3C01
literature	D3C02
stories	D3C03
food	D3C04

Step 3: Merge and alphabetize the list

- behaviors D2C04
- cats D1T01, D2T04, D1C01, D2C01
- children D1C03, D3C01
- dogs D1C02, D1T02, D2T05, D3T03, D2C02
- enemies D1T04
- feeding D2T03, D2C03
- food D3C04
- hot D3T02
- literature D1C04, D3C02
- love D3T01
- methods D2T02
- misunderstood D1T06
- mortal D1T03
- new D2T01
- simply D1T05
- stories D3C03

Step 4: Query Evaluation

- Traverse lists for each query term
  OR: the union of component lists
  AND: an intersection of component lists
- Can use position for proximity
- Can also perform tf-idf style computations if look up weights for terms in the doc

Queries:

- cats AND feeding
  D2: Cat Term 1, 3; Text Term 4
- mortal OR love
  D3: Text Term 1
  D1: Text Term 3

Introduction To Anatomy of Google

- Most material from presentation "Anatomy of a Large-Scale Hypertextual Web Search Engine" by Sergey Brin and Lawrence Page (1997)
- Flash Back to 1997
  - Amount of information and new users inexperienced in the art of web research rapidly growing.
  - People often starting with :
    - High quality human maintained indices
      - Effectively but subjective, expensive to build and maintain, slow to improve, and can't cover all topics
    - Keyword matching
      - Return too many low quality matches
  - It's necessary to build a large scale search engine
    - Large especially heavy use
    - Provide higher quality search results

Introduction (cont.)

- Web Search Engines--Scaling UP:1994-2000
  - Technology should scale to keep up with growth of web.
  - In 1994(WWWW, World Wide Web Worm):
    - Had index of 110,000 pages.
    - Receive 1500 queries per day.
  - In 1997(WebCrawler):
    - Claim to index from 2 to 100 million web pages
    - Altavista handled 20 million queries per day
  - Forecast in 2000
    - Index of Web will contain over a billion document
    - Hundreds of millions of queries per day.
  - Address Quality and Scalability problem
  - In 9/2005
    - ~24 billion pages with over 15,000 servers
    - 0.34 per second for one query (2003)

Introduction (cont.)

- Google:Scaling with the Web
  - Common spelling of “Googol ” or 10^100
  - Creating scalable search engine may face some problem:
    - Fast crawling technology:gather pages and keep them up to date
    - Storage space:used efficiently to store indices and pages.
    - Index System:process hundreds of gigabytes of data efficiently
    - Handle queries quickly
  - Designed to scale well to extremely large data sets:
    - Efficient use of storage space to store the index.
    - Data structures are optimized for fast and efficient access.
    - Expect the cost to index and store may decline.

Design Goals

- Improved search quality
  - In 1994, complete search index is enough
  - In 1997, “Junk results” becomes serious problem
  - Number of pages increase, but user’s ability not.
  - We want to get high precision, even at expense of recall.
  - The use of hypertextual information can help improve search and other application.
  - Google makes use of both link structure and anchor text.

System Features

- PageRank: Use Link Structure
  - Created maps containing 518 million of hyperlinks
  - Considered two points:
    - Citation importance
    - People’s subjective idea of importance.
  - Page Rank is an excellent way to prioritize the results of web keyword search.
  - Calculation:
    - $PR(A)=(1-d)+d[PR(T_1)/C(T_1)+..+PR(T_n/C(T_n)]$
      - d usually sets 0.85
      - use a simple iterative algorithm, and correspond to the principal eigenvector of normalized link matrix.

System Features(cont.)

– Intuitive Justification for PageRank
  - Damping Factor:
    – d is called damping factor, it means the probability at each page the “random surfer”
    – Can only add the damping factor d to a single page or a group of pages, and this allows for personalization.
  - Citation Concept:
    – A page can have high page rank if there are many pages that point to it, or if there are some pages that point to it and have a high rank.

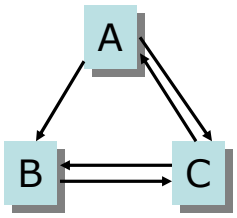
How PageRank is calculated ?

$$PR(A) = (1-d) + d\,(PR(T1)/C(T1) + \ldots + PR(Tn)/C(Tn))$$

- PR(A) is the PageRank of Page A
- d is a dampening factor(0-1). Nominally this is set to 0.85
- PR(T1) is the PageRank of a site pointing to Page A
- C(T1) is the number of links off that page
- PR(Tn)/C(Tn) means we do that for each page pointing to Page A
- Minimum: 1-d Maximum: (1-d)+dN

PageRank Calculator:
http://www.markhorrell.com/seo/pagerank.asp

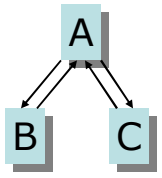
The Iterative Computation of PageRank



PR(A) = (1-0.85) + 0.85 (PR(C) / 2)
PR(B) = (1-0.85) + 0.85 (PR(A) / 2 + PR(C)/2)
PR(C) = (1-0.85) + 0.85 (PR(A) / 2 + PR(B)/1)
(PR : 1 d : 0.85)

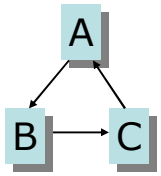
Iteration	PR(A)	PR(B)	PR(C)
0	1	1	1
1	0.15+0.85(1/2) = 0.575	0.15+0.85(1/2+1/2) =1	0.15+0.85(1/2+1) = 1.425
2	0.15+0.85(1.425/2) = 0.756	0.15+0.85(.575/2+1.425/2) =1	0.15+0.85(.575/2+1) = 1.244
...			
28	= 0.702	=1	=1.298245614
29	= 0.702	=1	=1.298245614

Internal Structures and Linkages



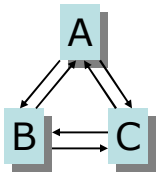
(Hierarchical)

Page A = 1.4594595
Page B = 0.7702703
Page C = 0.7702703



(Looping)

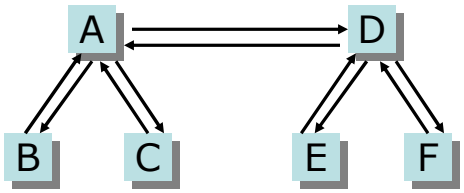
Page A = 1
Page B = 1
Page C = 1



(Interlinking)

Page A = 1
Page B = 1
Page C = 1

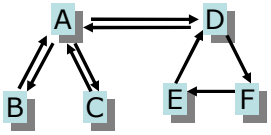
Link exchange(1)



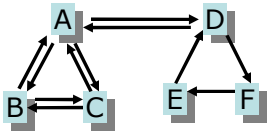
Page A = 1.4594595
Page B = 0.7702703
Page C = 0.7702703
Total PR: 3.0
Page A = 1.7234043
Page B = 0.6382979
Page C = 0.6382979
Total PR: 3.0

Page D = 1.4594595
Page E = 0.7702703
Page F = 0.7702703
Total PR: 3.0
Page D = 1.7234043
Page E = 0.6382979
Page F = 0.6382979
Total PR: 3.0

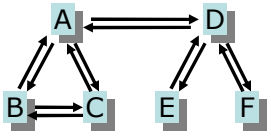
Link exchange(2)



Page A = 1.8623058 Page D = 1.3183416
Page B = 0.6776533 Page E = 0.7537509
Page C = 0.6776533 Page F = 0.7102952
Total PR: 3.2176124 Total PR: 2.7823877



Page A = 1.4934575 Page D = 1.1675242
Page B = 0.9967762 Page E = 0.6992681
Page C = 0.9967762 Page F = 0.6461978
Total PR: 3.4870099 Total PR: 2.5129901



Page A = 1.3913813 Page D = 1.5419126
Page B = 0.9464778 Page E = 0.5868752
Page C = 0.9464778 Page F = 0.5868752
Total PR: 3.2843369 Total PR: 2.7156630

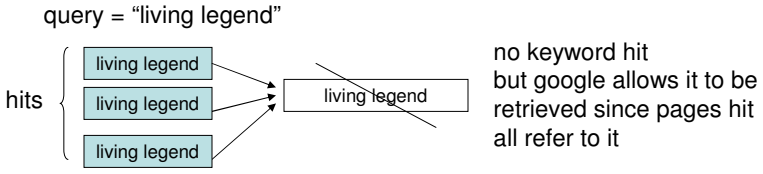
Penalty

Google wants to penalize a page--it is assigned a PageRank of zero.

- Spam(i.e., excessive repetition of keywords, same color text as background, deceptive or misleading links)
- Link farms(Reciprocal Link)

In Practice

- PageLink enhanced to allow retrieval of pages that are ranked highly if search keywords found on citation pages but not on target page



- Googlebomb
  - A user registers many domains and all of them link to their main site to influence google pagerank
  - Adam Mathes made “talentless hack” refer to his friend Andy Pressman; “more evil than Satan” refer to Microsoft
  - Ends when/if news media picks up and writes stories

System Features(cont.)

- Anchor Text
  - The text of links is treated in a special way.
  - We associate anchor text with the page the link points to
    - It often provides more accurate descriptions of web pages
    - Exists when pages can't be indexed by text-based search engine
  - In 24 million pages, we had over 259 million anchors which we indexed.

System Features(cont.)

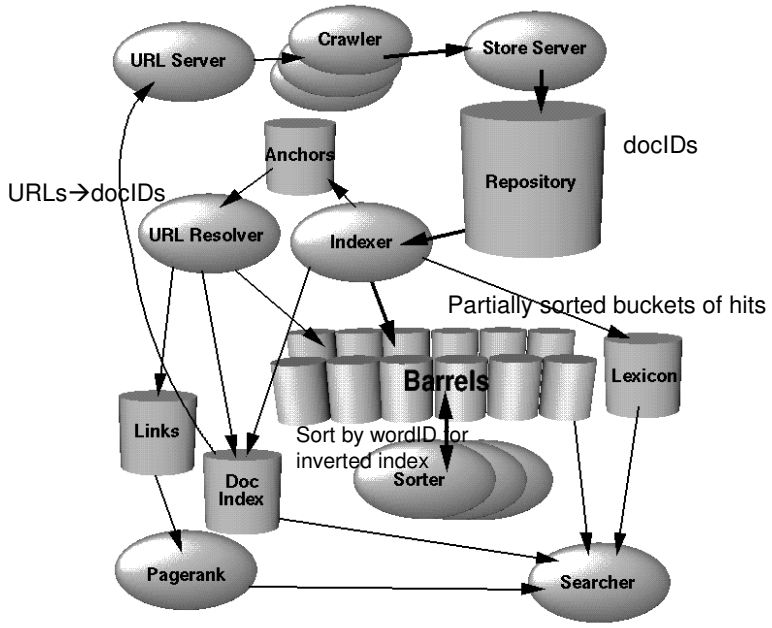
- Other features:
  - It has location information for all hits and so it makes extensive use of proximity in search.
  - Keep track of some visual presentation details
  - Full raw HTML of pages is available in a repository.

Related Work

- Information Retrieval
  - IR system is on small well controlled homogeneous collections such as scientific papers or news.
    - TREC takes 20GB as their benchmark compared to 147GB.
  - Things that work well on TREC don't produce good results on the web
    - Vector model will return very short document that are the query plus a few word
      - ex.Bill Clinton=>Bill Clinton sucks
    - They claimed that users should specify more accurately query.
  - We claim that user can get reasonable number and quality results for any precision or simple queries.

System Anatomy

- Google Architecture Overview:Figure 1
- Major Data Structure
  - Big Files:Virtual files spanning multiple file system
  - Repository:figure 2
    - use zlib (3:1) to compress
  - Document Index:Keep information about each documents
    - Add link to link list
    - Convert URLs into docIDs
  - Lexicon: figure 3
  - Hit Lists:a list of occurrences of a particular word in a particular document



Repository: 53.5 GB = 147.8 GB uncompressed

sync	length	compressed packet
sync	length	compressed packet

...

Packet (stored compressed in repository)

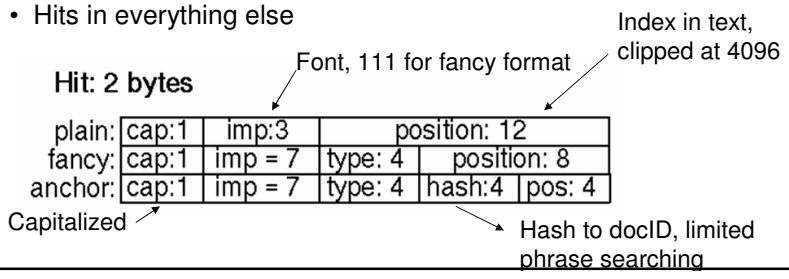
docid	ecode	urllen	pagelen	url	page

Document Index

- The document index keeps information about each document. It is a fixed width ISAM (Index sequential access mode) index, ordered by docID.
  - Document status
  - Pointer to the repository
  - Checksum
  - Document statistics
- File to convert URLs into docIDs
  - List of URL checksums with corresponding docIDs sorted by checksum
  - To find the docID of an URL, the URL's checksum is computed and binary search performed on checksum file

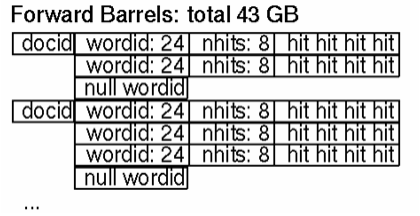
Hit List

- A hit list corresponds to a list of occurrences of a particular word in a particular document including position, font, and capitalization
- Takes up most of the space in the inverted index
- Compact encoding uses two bytes for every hit
  - Fancy hits
    - Hits in an URL, title, anchor text, or meta tag
  - Plain hits
    - Hits in everything else



Forward Barrels

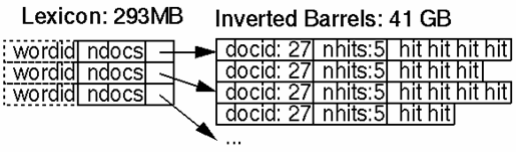
- Barrels of documents and list of hits for that document
  - Used 64 barrels



Partially sorted – Barrels assigned a range of wordid's.
This means that one docid may be replicated in multiple barrels.

Inverted Index

- The inverted index consists of the same barrels as the forward index, except that they have been processed by the sorter.
  - For every valid wordID, the lexicon contains a pointer into the barrel that wordID falls into. It points to a doclist of docID's together with their corresponding hit lists.
  - This doclist represents all the occurrences of that word in all documents.



- Actually keep two sets of inverted barrels
  - Small barrels set include title or anchor hits and larger set for all hit lists.
  - Check the small set of barrels first and if there are not enough matches within those barrels we check the larger ones.

System Anatomy(cont.)

- Indexing the Web
  - Parsing
  - Indexing Documents into Barrels
  - Sorting
- Searching
  - The goal of searching is to provide quality search results efficiently.
  - Query Evaluation, figure 4
  - To put a limit on response time, only found 40,000 pages, so sub-optimal results may be returned.

Figure 4

1.Parse the query.
2.Convert words into wordIDs.
3.Seek to the start of the doclist in the short barrel for every word.
4.Scan through the doclists until there is a document that matches all the search terms.
5.Compute the rank of that document for the query.
6.If we are in the short barrels and at the end of any doclist, seek to the start of the doclist in the full barrel for every word and go to step 4.
7.If we are not at the end of any doclist go to step 4.
Sort the documents that have matched by rank and return the top k.

Figure 4. Google Query Evaluation

System Anatomy(cont.)

- The Ranking System
  - Google maintains much more information about web documents
    - Hitlist includes position, font, and capitalization
    - Anchor text
  - For single word search:
    - Give different type (ex.title, URL, anchor) different weight
    - Count the word weight and sum to IR score
  - For multi word search:
    - Besides type, the hits occurs close together are weighted higher than hits occur far apart.
  - Feedback:Use users' feedback to judge the weight parameter in the system

Results

- Google produce better results than commercial search engine.
  - Figure 5
  - The result show the feature of Google
    - Good result quality and no broken links
    - Anchor Text based(ex. E-mail can be shown)
    - No results about a Bill other than Clinton or about a Clinton other than Bill
  - Storage Requirement:Table1

Query: bill clinton

http://www.whitehouse.gov/

100.00% (no date) (0K)

http://www.whitehouse.gov/

Office of the President

99.67% (Dec 23 1996) (2K)

http://www.whitehouse.gov/WH/EOP/OP/html/OP_Home.html

Welcome To The White House

99.98% (Nov 09 1997) (5K)

http://www.whitehouse.gov/WH/Welcome.html

Send Electronic Mail to the President

99.86% (Jul 14 1997) (5K)

http://www.whitehouse.gov/WH/Mail/html/Mail_President.html

mailto:president@whitehouse.gov

99.98%

mailto:President@whitehouse.gov

99.27%

The "Unofficial" Bill Clinton

94.06% (Nov 11 1997) (14K)

http://zpub.com/un/un-bc.html

Bill Clinton Meets The Shrinks

86.27% (Jun 29 1997) (63K)

http://zpub.com/un/un-bc9.htm

President Bill Clinton - The Dark Side

97.27% (Nov 10 1997) (15K)

http://www.realchange.org/clinton.htm

$3 Bill Clinton

94.73% (no date) (4K)

http://www.gatewy.net/~tjohnson/clinton1.html

Figure 5. Sample Results from Google

Table 1

Storage Statistics	
Total Size of Fetched Pages	147.8 GB
Compressed Repository	53.5 GB
Short Inverted Index	4.1 GB
Full Inverted Index	37.2 GB
Lexicon	293 MB
Temporary Anchor Data (not in total)	6.6 GB
Document Index Incl. Variable Width Data	9.7 GB
Links Database	3.9 GB
Total Without Repository	55.2 GB
Total With Repository	108.7gb
Web Page Statistics	
Number of Web Pages Fetched	24 million
Number of Urls Seen	76.5 million
Number of Email Addresses	1.7 million
Number of 404's	1.6 million

Performance

- System performance:9 days to download 26 million pages(worse case)
- Search Performance:
  - query time range from 1 to 10 sec.
  - Time is mostly dominated by disk IO over NFS
  - Table 2

Table 2

	Initial Query Same Query		Repeated (IO mostly cached)	
Query	CPU Time(s)	Total Time(s)	CPU Time(s)	Total Time(s)
al gore	0.09	2.13	0.06	0.06
vice president	1.77	3.84	1.66	1.80
hard disks	0.25	4.86	0.20	0.24
search engines	1.31	9.63	1.16	1.16

Conclusion

- Google has some features:
  - Is suited for large-scale web environment.
  - Can provide high quality results.
- Future Work
  - Intelligent update algorithm
  - Other scalable methods
  - Other weighting techniques
    - Add weight to page in bookmarks in order to construct personal web