Fast Generation of Result Snippets in Web Search

Turpin, A., Tsegay, Y., Hawking, D. and Williams, H.E.

In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 127-134). ACM, 2007

Document Compaction for Efficient Query Biased Snippet Generation

Tsegay, Y., Puglisi, S.J., Turpin, A. and Zobel, J.

In *European Conference on Information Retrieval* (pp. 509-520). 2009

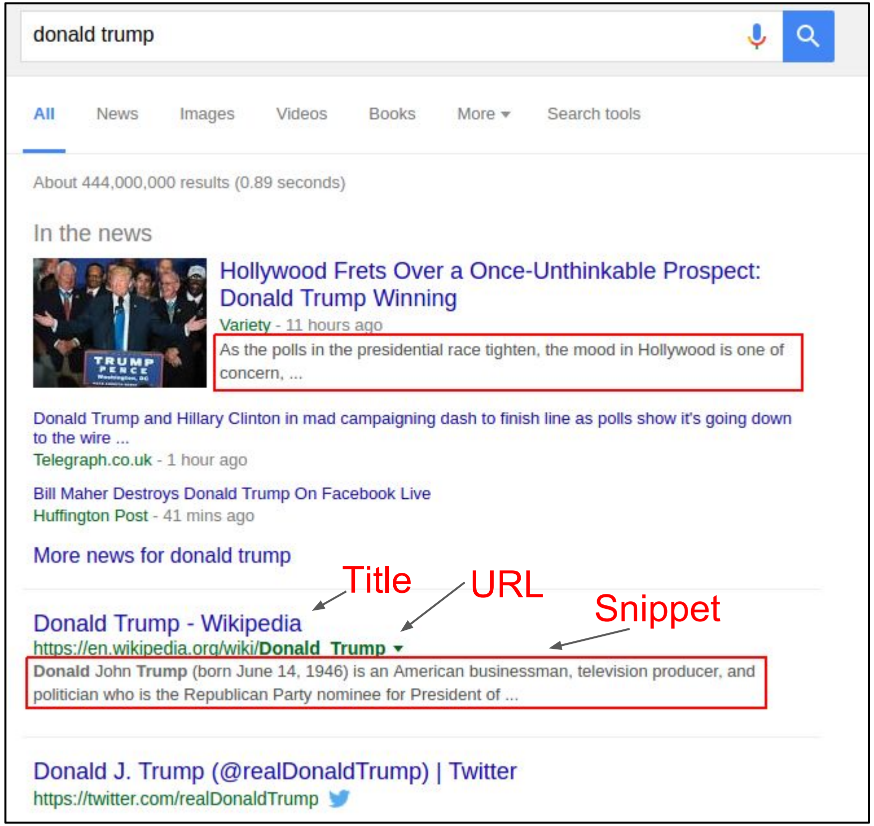
Presented by Erika Siregar
CS 834 - Introduction to Information Retrieval
Fall 2016
Old Dominion University
November 3, 2016

What is Snippet?

- Short fragment extracted from document
- A sneak preview of the document contents.

Two types of Snippet:

- Static
- Query-biased



How snippet is generated?

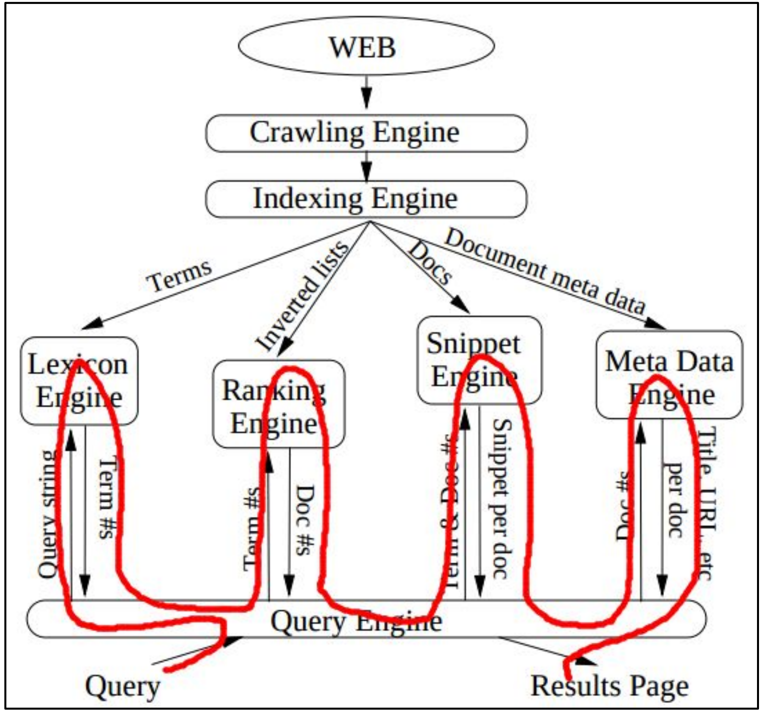


Fig. 1 Abstraction of some subsystems in search engine

1. **Lexicon engine** → **maps** query terms to integers
2. **Ranking engine** → **retrieves** inverted lists for each term and use them to get a **ranked list of documents**
3. **Snippet engine** → use document numbers & query term numbers to generate snippets.
4. **Metadata engine** → fetches information (title, URL, etc)

So, What is The Problem?

Query-biased Snippet is dynamic.

Deal with these ...

- Storage → Order of ten billion web pages
- Load → Hundreds of millions of search queries per day
- Response → File accessing is a major bottleneck

users

we don’t want to waste our time waiting for the snippet

Search engine

Identify the running time ...

Majority of **time spent** generating a snippet is in **locating the document on disk (seek)**: **64%** for whole documents.

With **1% of documents cached** → around **80% of disk seeks are avoided**

How to speed up?

- Utilize the in-memory **Cache**
  - Reducing Disk Access

- Document **Compression**
  - The smaller document size, the more documents can be cached

- Document **Compaction**
  - Consider only the important part

Caching

What should be cached?

- **Document cached** → Frequently accessed documents
- **Query cached** → Precomputed result pages for popular queries

Document Compression

Compression is processed by **Snippet Engine**

- Proposed Compressor → **Compressed Token System (CTS)**
- Well-known Compressor (baseline) → Zlib Compressor Library

How CTS Works?

1. **Parsing** the document
   a. Remove tags
   b. Determine the beginning and ending of a sentence
      i. Restrictions:
         5 <= sentence words <= 20
   c. Output: word tokens & non-word tokens (punctuations)

Word Model

Code	Word
0	"with"
1	"you"
2	"how"
3	"are"
4	"in"
5	"computer"
. . .	. . .

Non-Word Model

Code	Non-word
0	" "
1	"."
2	", "
3	" – "
4	"""
5	";"

2. **Pruning** algorithm
   a. **Collect** the words and non-words tokens
   b. Construct **model** → estimate distribution of the term frequencies
      i. Discard rare tokens (low frequency) → save storage
   c. **Encode** using vbyte scheme → higher frequency = smaller bits
      i. **Replace** the tokens with their vbyte codes
   d. **Discarded tokens** are replaced with **ESCAPE** symbol.

- If the probability of symbol 'u' is estimated to be 2%, the corresponding information content is 5.6 bits
- if 'u' happens to be the next symbol that is to be coded, it should be transmitted in 5.6 bits

Experiment

- **Data:** WT100G, WT10G, WT50G

- Queries from **Excite logs** (http://msxml.excite.com/) in 1997
    - All queries are stemmed, stopped, and sorted alphabetically.

- Use **Zettair** search engine → get the list of documents for each query.

- Okapi **BM25** → document rank
    - Score for each document is independent of any query.

Is CTS Better Than Zlib?

- In compression, **CTS loss over zlib**, both reduce about **20%** of the original size

	WT10G	WT50G	WT100G
No. Docs. ($\times 10^6$)	1.7	10.1	18.5
Raw Text	10,522	56,684	102,833
Baseline(*zlib*)	2,568 (24%)	10,940 (19%)	19,252 (19%)
CTS	2,722 (26%)	12,010 (21%)	22,269 (22%)

Table 1: Total storage space (Mb) for documents for the three test collections both compressed, and uncompressed.

- But, in snippet generation time, CTS performs about **50%** better than Zlib

	WT10G	WT50G	WT100G
Baseline	75	157	183
CTS	38	70	77
Reduction in time	49%	56%	58%

Table 2: Average time (msec) for the final 7000 queries in the Excite logs using the baseline and CTS

Compaction

- **Reorder sentences by significance → Consider Only The Important Part**

- The **techniques**:
  - **Natural order** → First sentence should introduce paragraph. Happened in **Well Authored Document.**
  - **Significant terms (ST)** → Score based on term frequency

$$f_{d,t} \geq \begin{cases} 7 - 0.1 \times (25 - s_d), & \text{if } s_d < 25 \\ 7, & \text{if } 25 \leq s_d \leq 40 \\ 7 + 0.1 \times (s_d - 40), & \text{otherwise,} \end{cases}$$

  - **Query log based (QLt)** → Score based on **past query terms**. **Move** sentence with many past query term **to front.**
  - **Query log based (QLu)** → Same as QLt, but considers only **unique terms**

Sentence Ranking

IN A document broken into one sentence per line, and a sequence of query terms.

1 For each line of the text, $\mathcal{L} = [w_1, w_2, \ldots, w_m]$

2 Let h be 1 if $\mathcal{L}$ is a heading, 0 otherwise.

3 Let ℓ be 2 if $\mathcal{L}$ is the first line of a document, 1 if it is the second line, 0 otherwise.

4 Let c be the number of w_i that are query terms, counting repetitions.

5 Let d be the number of distinct query terms that match some w_i.

6 Identify the longest contiguous run of query terms in $\mathcal{L}$, say $w_j \ldots w_{j+k}$.

7 Use a weighted combination of c, d, k, h and ℓ to derive a score s.

8 Insert $\mathcal{L}$ into a max-heap using s as the key.

OUT Remove the number of sentences required from the heap to form the summary.

Figure 2: Simple sentence ranker that operates on raw text with one sentence per line

- Document is broken into sentences S where S = [w1, w2, ...,wm].
- Query Q where Q = {q1, q2, ..., qn}

Notes:

h → sentence is a heading

l → sentence is first or second line of document

k → length of longest contiguous run of qi in S

c → count of wj ∈ Q

d → c minus repititions

Which Technique is Better?

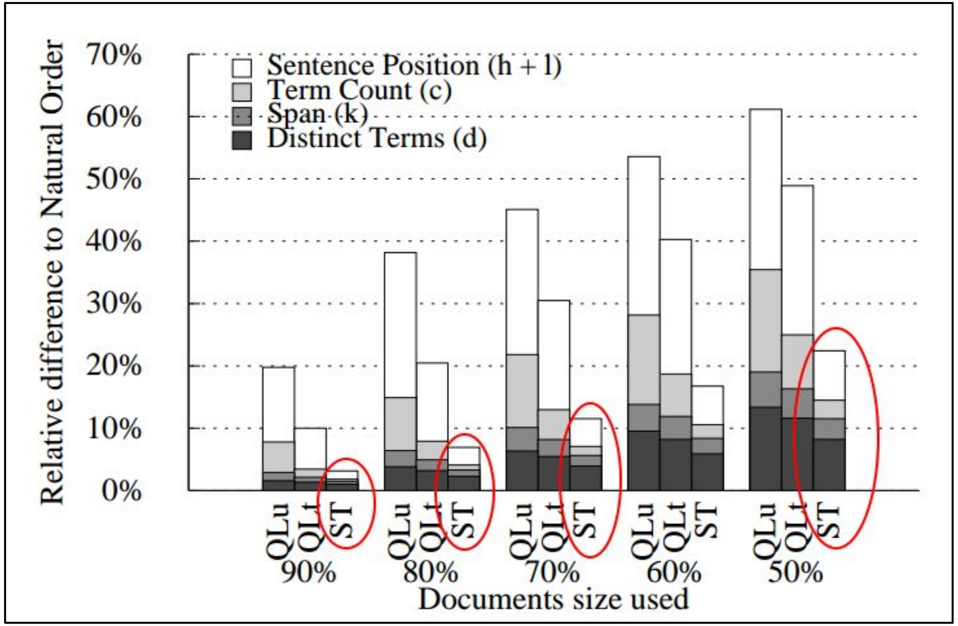


Figure 6: Relative difference in the snippet score components compared to Natural Ordered documents when the amount of documents processed is reduced, and the sentences in the document are reordered using Query Logs (QLt, QLu) or Significant Terms (ST).

- **Significant Terms (ST) wins.**
- The greatest change over all methods is in the **sentence position (h + l) component** of the score

Could we have a better document compaction method?

Let’s go to the paper 2 …

Why focus on document compaction?

- Reduce disk seeks and reads.
- smaller size → more documents in the cache
- Fewer sentence to evaluate during scoring phase.

Problem with the previous (CTS):

CTS relies on query logs, which has 2 shortcomings:

1. The drift of the query natures must be followed by the change of the document representation → **computational overhead**
2. The presence of a suitable query log is assumed.

New idea...

New compaction approach

1. Reordering the sentences based on their scores
2. Pruned the documents to the desired length
3. Use this pruned version as the surrogates of the full document.

How to reorder?

Investigating 2 methods:

1. TF.IDF weighting

$$\mathrm{TF} = 1 + \ln(f_{d,t}),\ \ \mathrm{IDF} = \ln\left(\frac{N}{d_f}\right)$$

2. Kullback-Leibler Divergence

$$\mathrm{KLD}(t,d,c) = (P(t|d))^{1-\delta}\left(\log\frac{P(t|d)}{P(t|c)}\right)^{1+\delta}$$

How to prune?

Considering 3 methods:

1. Fixed percentage of sentences → **used**
2. Fixed number of sentences
3. Threshold T

Experiment

- Data from TREC Corpus → WT10G, WT100G
- Excite query logs

Generating the snippets

1. Parsing
2. Tokenizing → words and non-words
3. Identifying sentences
    a. End sentence marker → **. ? !**
    b. HTML tags → <h1>, </h1>, <p>, </p>, and
    c. Length threshold → max. 30 words
4. Scoring the sentences
    a. Simplify Turpin's method
    b. Sort sentence by:
        i. Primary key → count of unique query terms in the sentence
        ii. Secondary key → longest span of query terms in the sentence
    c. Replace Turpin's tertiary key (total count of query terms in a sentence) with TFIDF
5. Remove duplicate sentences
    a. Duplicate = 80% of same contents with n-gram 5

Evaluation

Snippet generated from pruned document

Vs

Snippet generated from full document.

Results

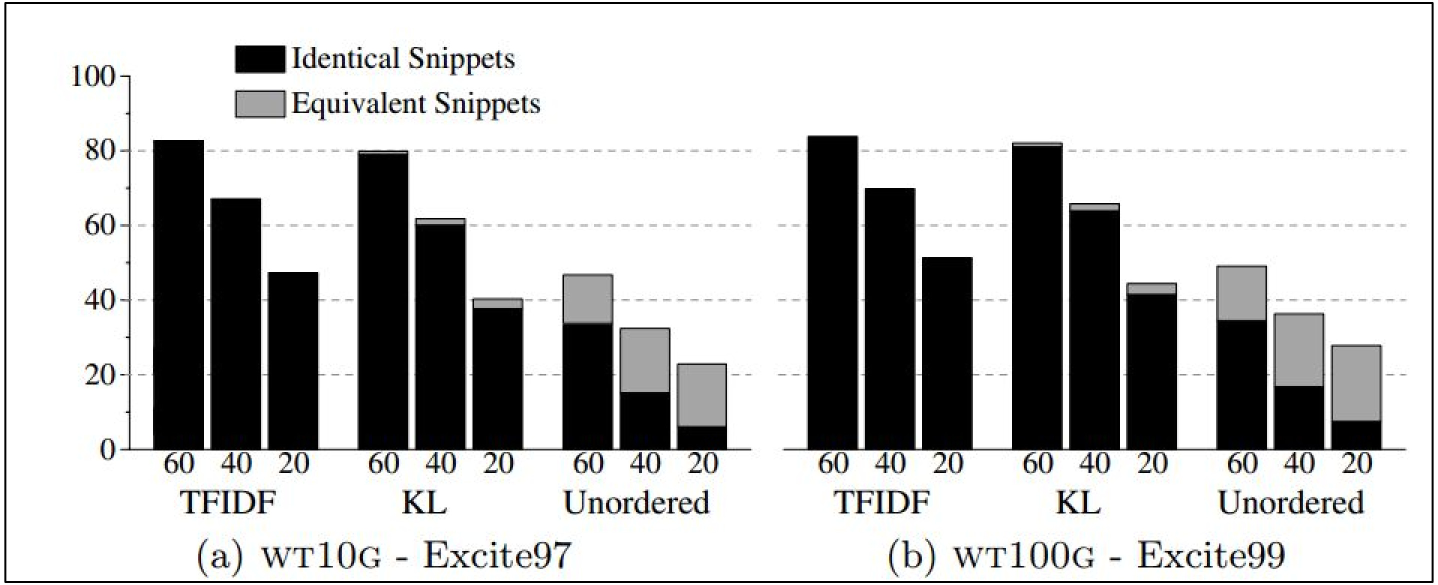


Fig. 1. Quality of snippets generated from wt10g for Topics 451-550. Documents were reordered using the various reordering schemes and then pruned. The dark bars indicate the percentage of pruned documents with identical snippets to the full documents while the light colored bar indicates those that generated snippets with the same QBS score.

How to ensure that this is a good snippet?

- What if there is any degradation?

- Simple-go-back (SGB) approach:
  - Check if the snippet contains sentences with no query term.
  - Check if the number of query terms in snippet $<$ number of query terms in the index.
  - If TRUE $\rightarrow$ sentences contain query terms were removed.
  - Generate snippet from the full document.

SGB vs No SGB results

Collection–queries	Identical Snipp. (%)		Size (%)	
	No SGB	SGB	No SGB	SGB
WT10G–TOP451-550	69.32	89.39	41.48	62.58
WT10G–Excite97	66.89	82.63	40.78	51.93
WT100G–Excite99	69.49	80.74	40.62	47.71

Table 2. Percentage of snippets produced that are identical to those produced with whole documents, and the amount of data (percentage of whole documents) processed per snippet, with and without SGB

Conclusion

- Paper 1:
  - Devise new approach for faster snippet generation:
    - Cache
    - Compression → Compressed Token System (CTS)
    - Compaction
- Paper 2:
  - Improve the compaction method used by CTS
  - Generating snippet from pruned document.
  - Over 80% identical snippets.