US Patent 6,658,423

William Pugh

Detecting duplicate and near - duplicate files

- Worked on this problem at Google in summer of 2000
- I have no information whether this is currently being used
- I know that other people at Google were exploring other approaches
- I'll only discuss background work and what could be discovered from the patent application

Reasons for confidentiality

- Competitors (e.g., Microsoft)
- Search Engine Optimizers

Problem

- In a web crawl, many duplicate and near duplicate web pages are encountered
  - one study suggested than 30+% are dups
- Multiple URL's for same page
  - http://www.cs.umd.edu/~pugh vs http://www.cs.umd.edu/users/pugh
- Same web site hosted on multiple host names
- Web spammers

Why near - duplicate?

- Identical web pages are easy to explain, and easy to cope with
  - just use a hash function on web pages
- Near duplicate web pages typically arise from:
  - small amounts of dynamic content
  - web spammers

Obvious O(n^2) algorithm

- We could compare each pair of web pages and compute edit distance
- Could do this at time query result is generated

What would we do with the information?

- Identify mirrors or replicated web sites
- Avoid storing near-duplicate copies
- Avoid returning near-duplicate web pages in results
- Use it to improve page rank calculations

First approach

- Break a document up into chunks (sentences, paragraphs, shingles, ...)
- Fingerprint each chunk
- Two documents are similar if a large percentage of their fingerprints are in common
- Still have lots of data to process
- Iceberg query, hard to perform

Broder’s approach

- Andrei Broder of Digital/Compaq/Altavista had a number of papers on this problem

Shingles

- A *k*-shingle is a sequence of *k* consecutive words
  - The quick brown
  - quick brown fox
  - brown fox jumped
  - fox jumped over
  - ...

Resemblance

- Let S(A) be the shingles contained in A
  - or the 64-bit hashes of the shingles contained in A
- Resemblance of A and B given by

$$\frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|}$$

Sampling minima

- Let σ be a random permutation/hash function

$$\mathrm{Prob}[\min(\sigma(S(A))) = \min(\sigma(S(B)))] = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|}$$

First Implementation

- Choose a set of t random min-wise independent permutations
- For each document, keep a sketch of the t minima shingles (samples)
- Estimate similarity by counting common samples

SuperShingles

- Divide samples into *k* groups of *s* samples ($t = k*s$)
- Fingerprint each group => feature
- Two documents are considered near-duplicates if they have more than *r* features in common

Sample values

- Looking of resemblance of 90%
- Sketch size = 48 , divide into 6 groups of 14 samples
- Need r=2 identical groups/features to be considered near duplicates

How does this work?

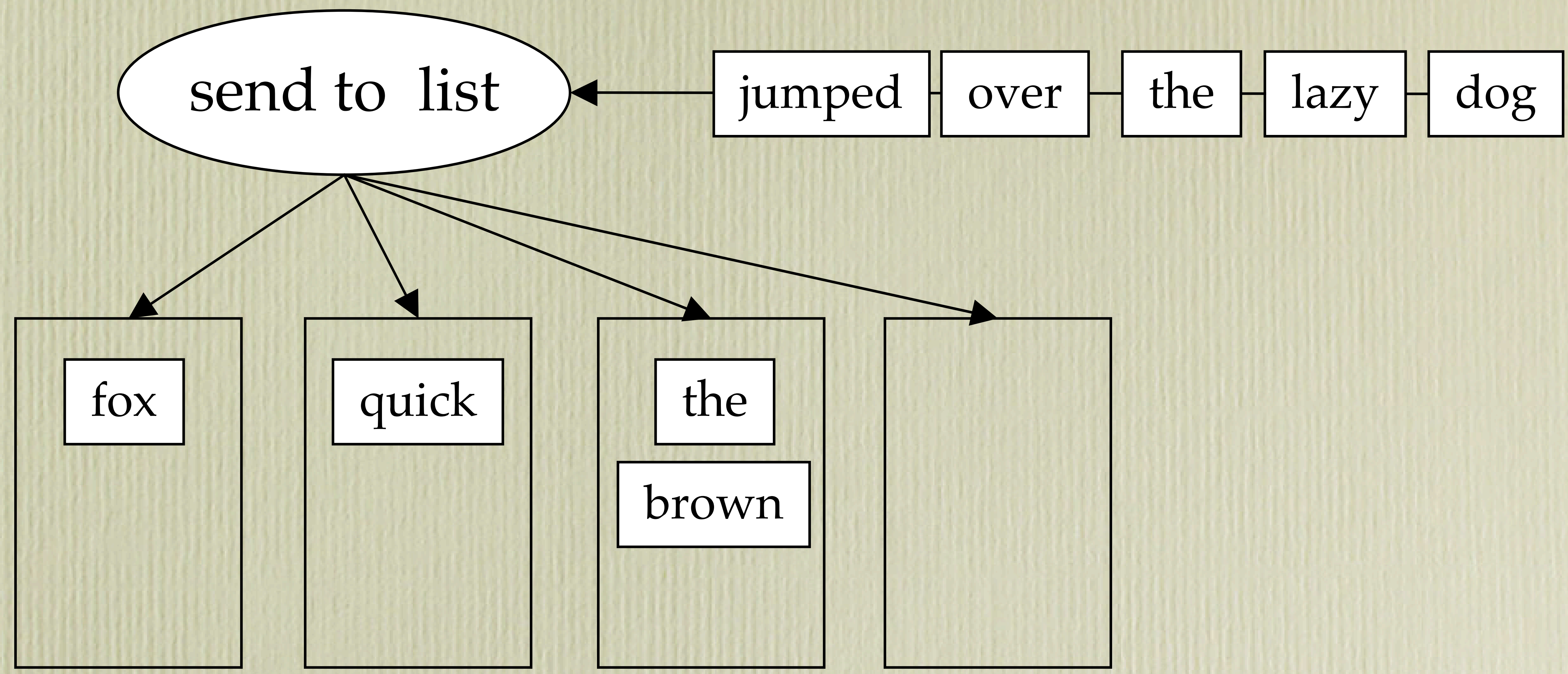
- Similarity model is OK, has good and bad features
- Can easily compare two documents to see if they are similar
- Expensive to find all similar document pairs

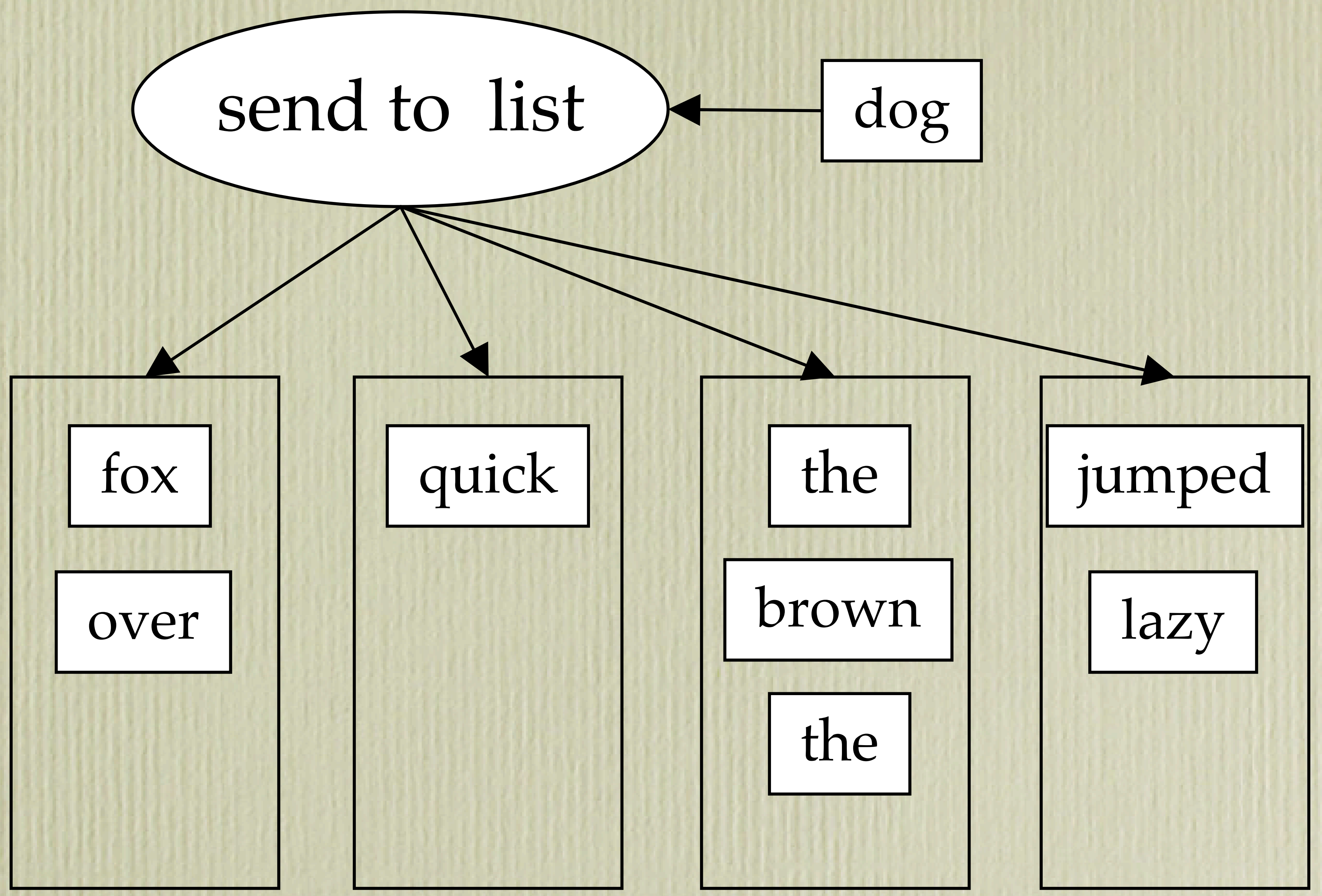
Finding all near - duplicate document pairs

- Want to find all document pairs that have more than *r* fingerprints in common
- Discard all fingerprints/features that occur in a single document
- If r > 1, we know have an iceburg query
    - lots of fingerprints that occur in two or more non-near-duplicate documents

US Patent 6,658,423

- Take the list of words (or shingles)
- Apply a hash function to each word
- Use hash value to determine which list it is appended to
- Compute fingerprint of each list
- Two documents are similar if they have any fingerprints in common


send to list
jumped
over
the
lazy
dog
fox
quick
the
brown

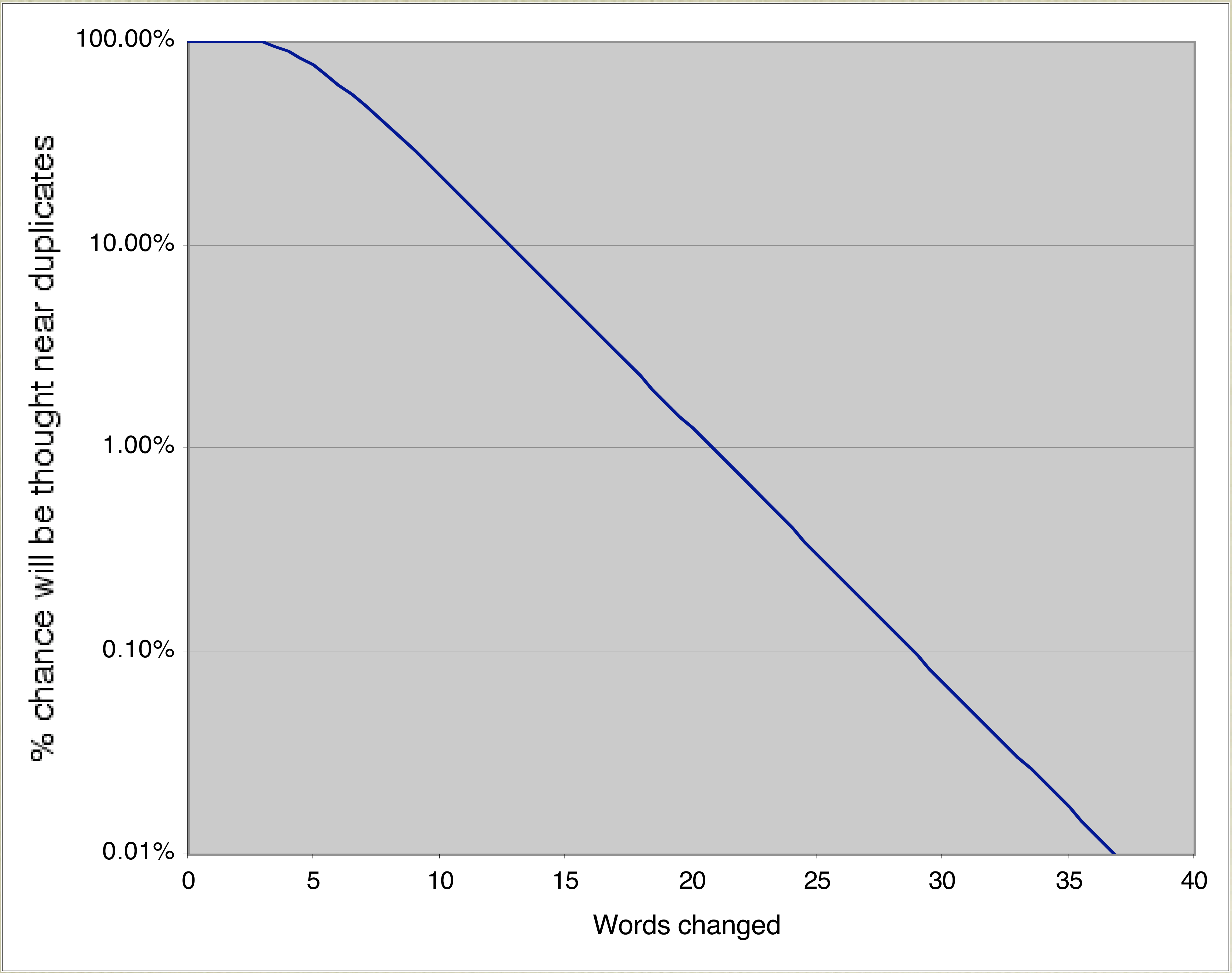

send to list
dog
fox
over
quick
the
brown
the
jumped
lazy

Similarity

- Different metric of similarity
- An edit consists of:
    - removing any or all occurrences of a word
    - adding any number of occurrences of a word
- Edit distance is minimum number of edits required to convert one document into another

Will two documents have a fingerprint in common?

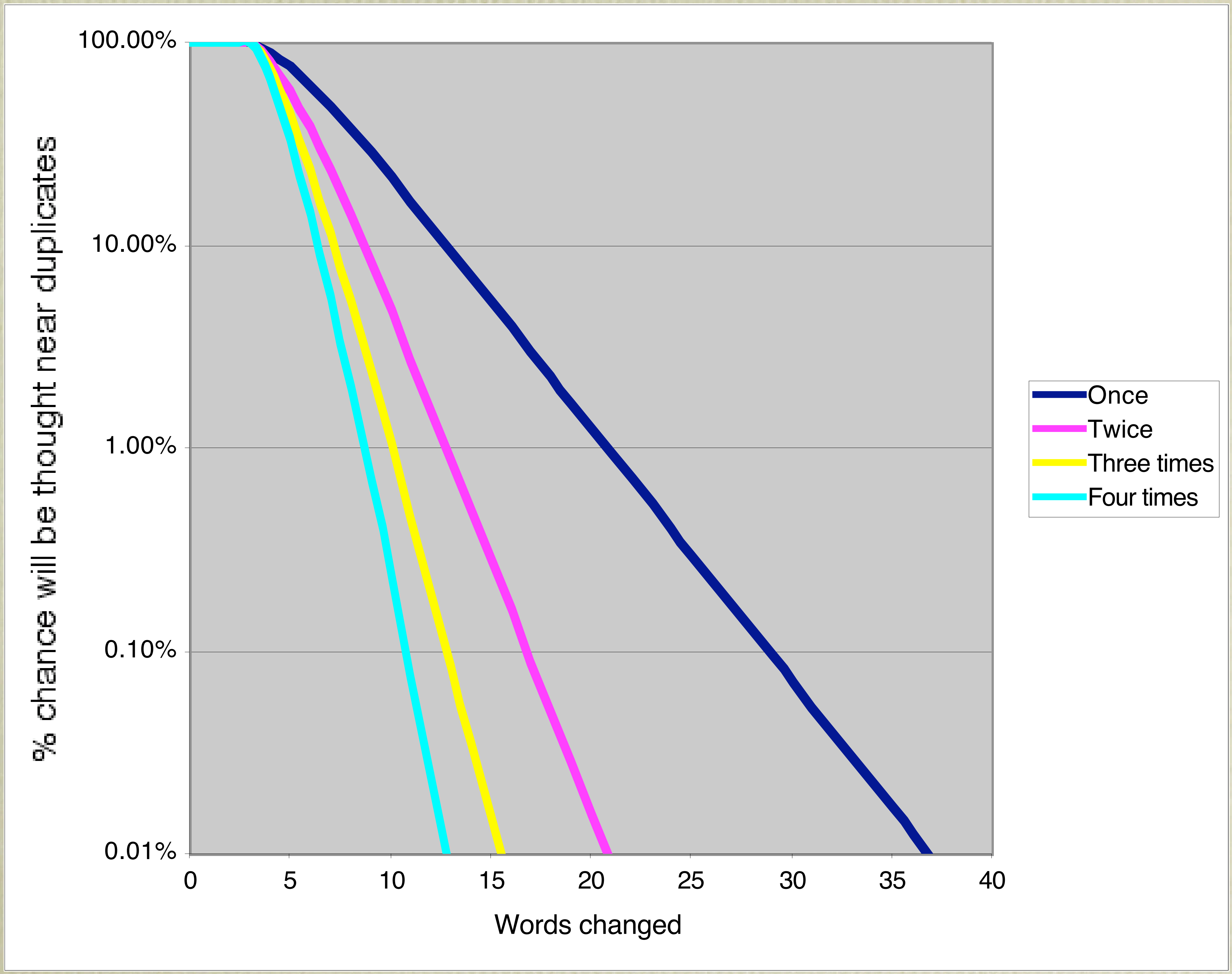
- Assume we use 4 lists/fingerprints per document
- Each edit will cause one randomly selected fingerprint to change
- How many changes are needed to create a high probability that all fingerprints have been changed?


100.00%
10.00%
1.00%
0.10%
0.01%
% chance will be thought near duplicates
0
5
10
15
20
25
30
35
40
Words changed

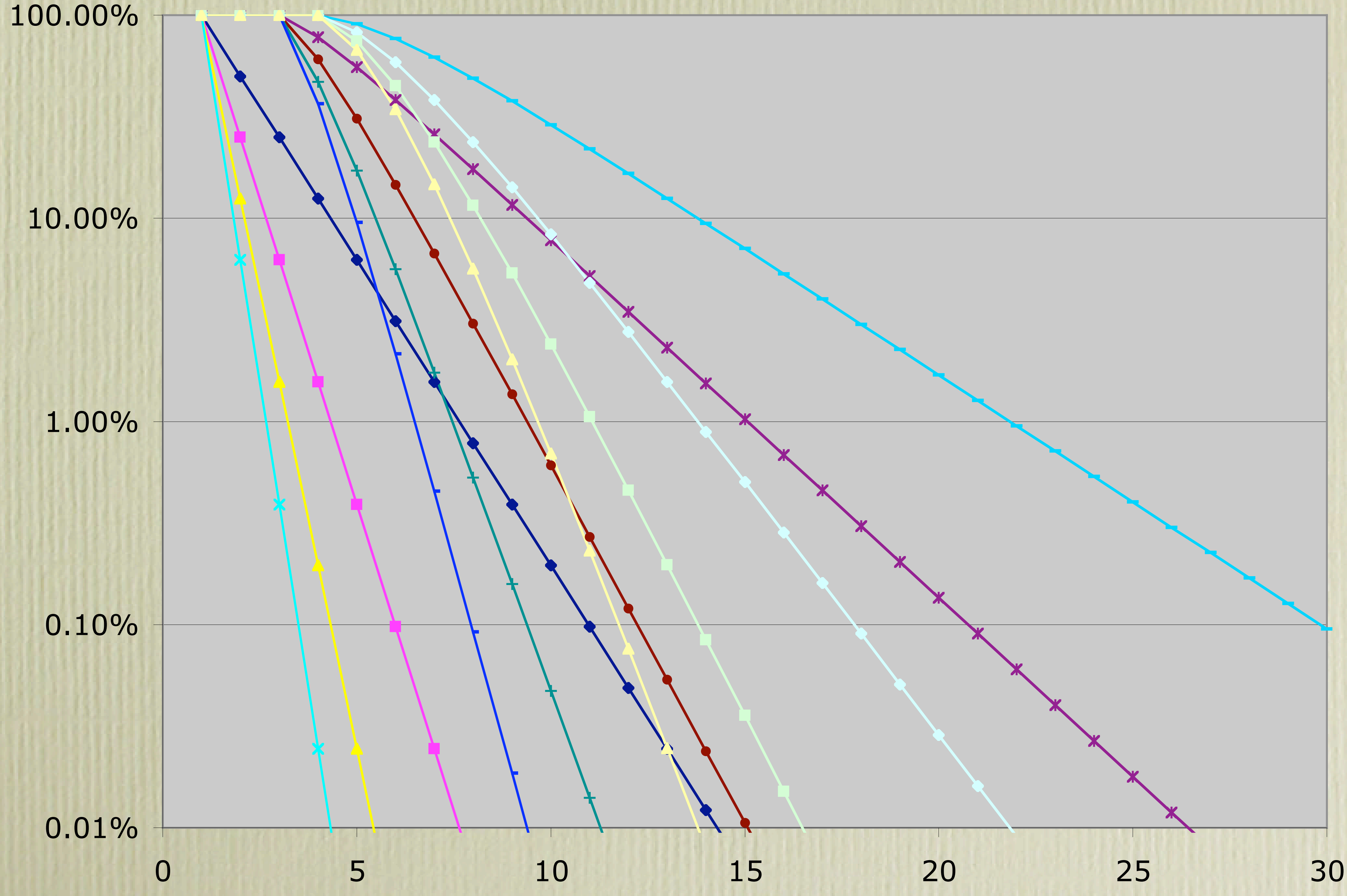
False Positive Rate

- 0.1% seems like a pretty low false positive rate
- unless you are indexing billions of web pages
- Need to be very careful about deciding to discard web pages from index
    - Less careful about eliminating near duplicates from query results

Can run test multiple times



Can vary # of lists and number of repeats



Discarding unique fingerprints

- Hash fingerprints into bitmaps to remove unique fingerprints
    - two bits per hash value (0, 1 or many)
    - first pass to count, second pass to discard
- Repeat passes with different hash function to handle collisions
- Partition fingerprints if we can't create bitmaps that are sparse enough

Finding clusters

- Sort fingerprint, docID pairs by fingerprint
- Perform union-find on docIDs
- Result is clusters of near-duplicate documents
  - with an assumption that similarity is transitive

Similarity Measures

- Replacement of one word with another counts as two changes
  - no matter how many replacements occur
- Moving a big chunk of text is a big change
  - unless you use a order-insensitive hash function
- Absolute diff, not % diff

Conclusion

- Unknown
- I don't know what Google has done with this idea
  - although they seem to be using something
- I haven't been able to talk with anyone else about this idea
  - until now