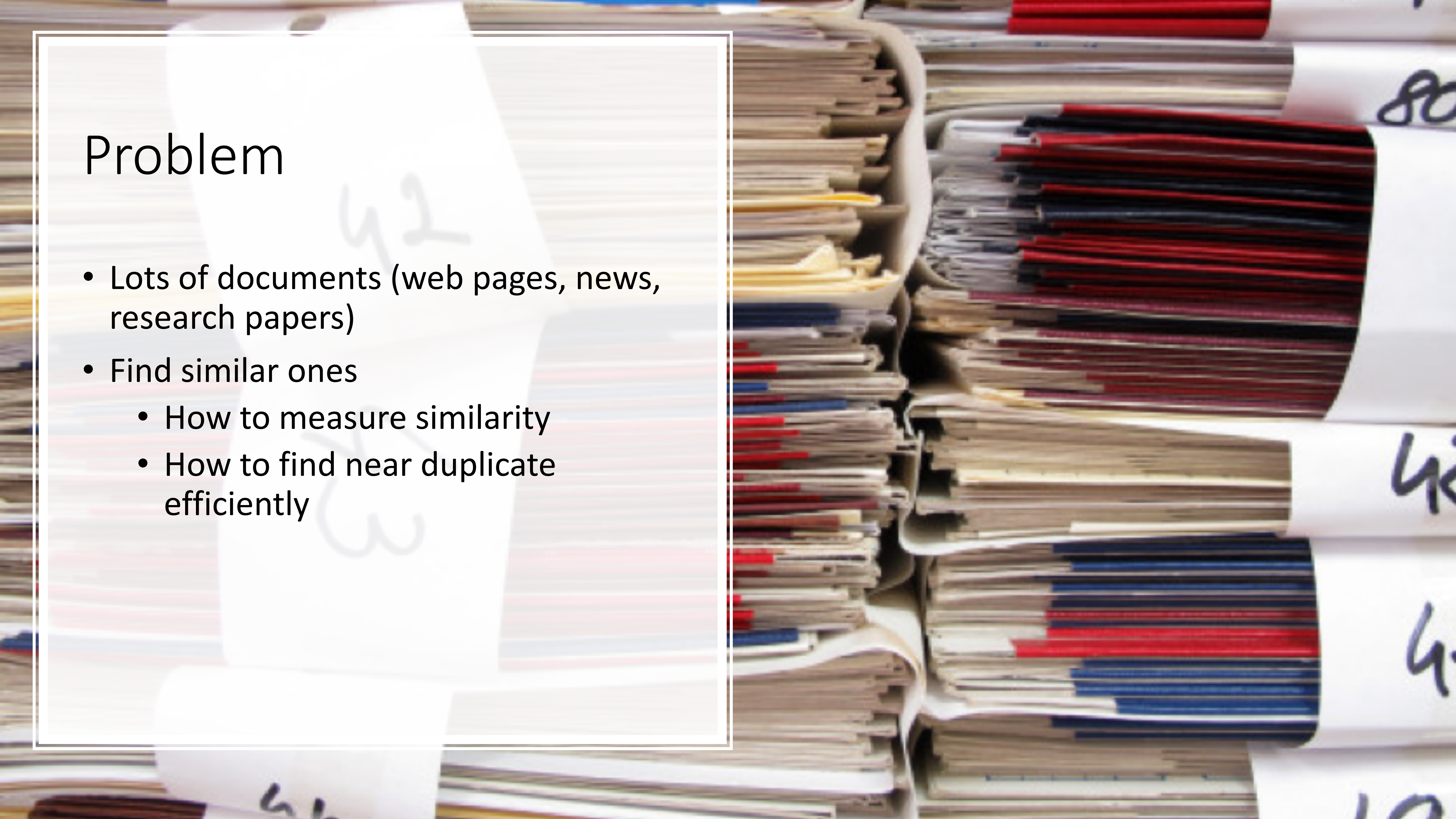
Near Duplicate Detection using Simhash

Presented by

Sumon Biswas

Problem

- Lots of documents (web pages, news, research papers)
- Find similar ones
  - How to measure similarity
  - How to find near duplicate efficiently



Applications

- Documents clustering
- Web mining
- Focused crawling
- Recommendation system
- Plagiarism checking
- Spam detection
- Computer vision

Outline

- Similarity measure
- Locality sensitive hashing
- Simhash algorithm
- Hamming distance problem

Similarity Measure

Two popular similarity measure between datasets:

Jaccard similarity:

$$sim(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

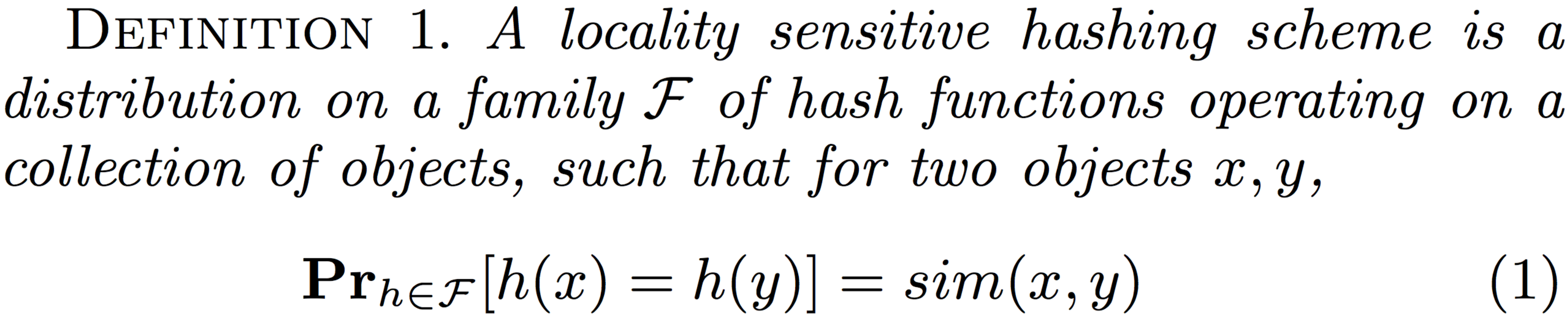
Cosine similarity:

$$sim(A,B) = \frac{|A \cap B|}{\sqrt{|A|\,|B|}}$$

Near Duplicate Detection

- Exact duplicate detection is relatively easy
    - Using checksum technique
- Near duplicate is more challenging
    - find all web pages that are at least 90% similar to web page *u* (e.g., more than 90% words are same)
- To search O(n) comparisons are required
- To find all similar pairs in a collection O(n^2) comparisons are required

Locality Sensitive Hashing

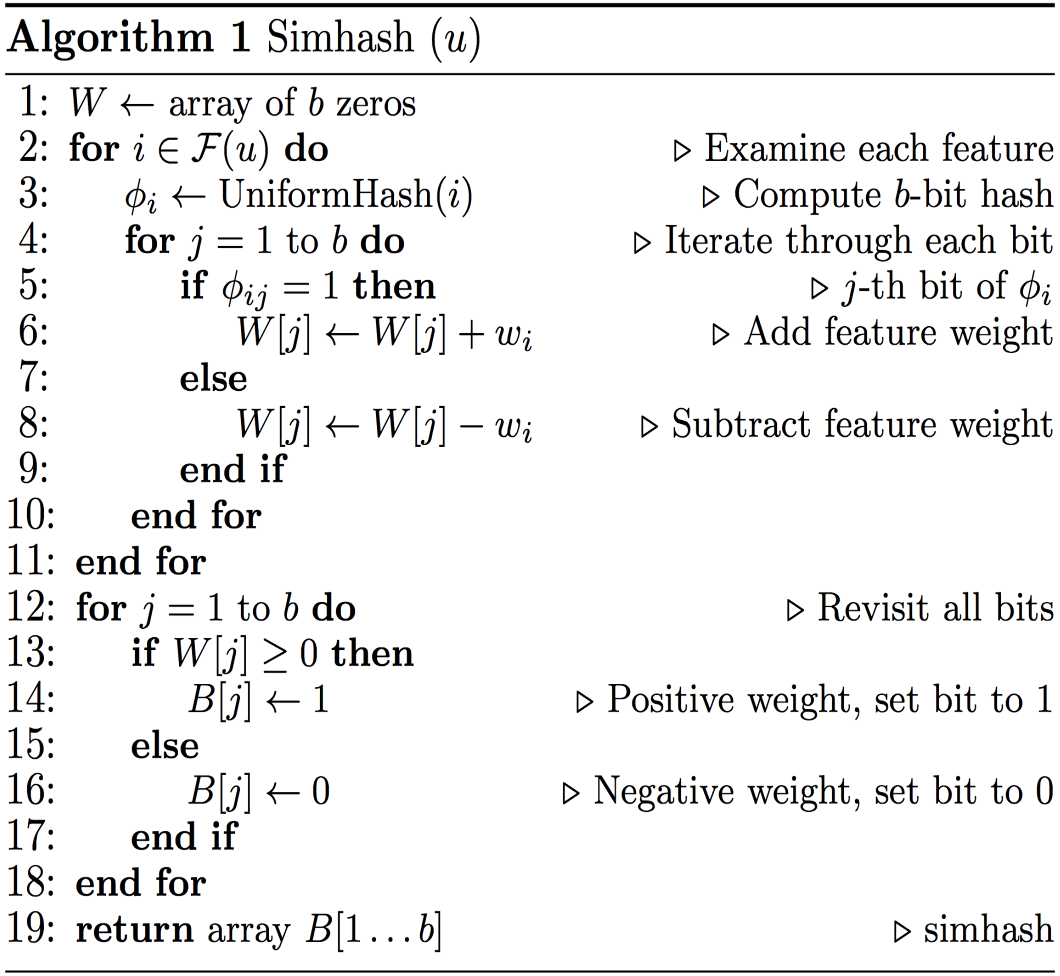


Definition 1. *A locality sensitive hashing scheme is a distribution on a family $\mathcal{F}$ of hash functions operating on a collection of objects, such that for two objects x, y,*

$$\mathbf{Pr}_{h\in\mathcal{F}}[h(x) = h(y)] = sim(x, y) \tag{1}$$

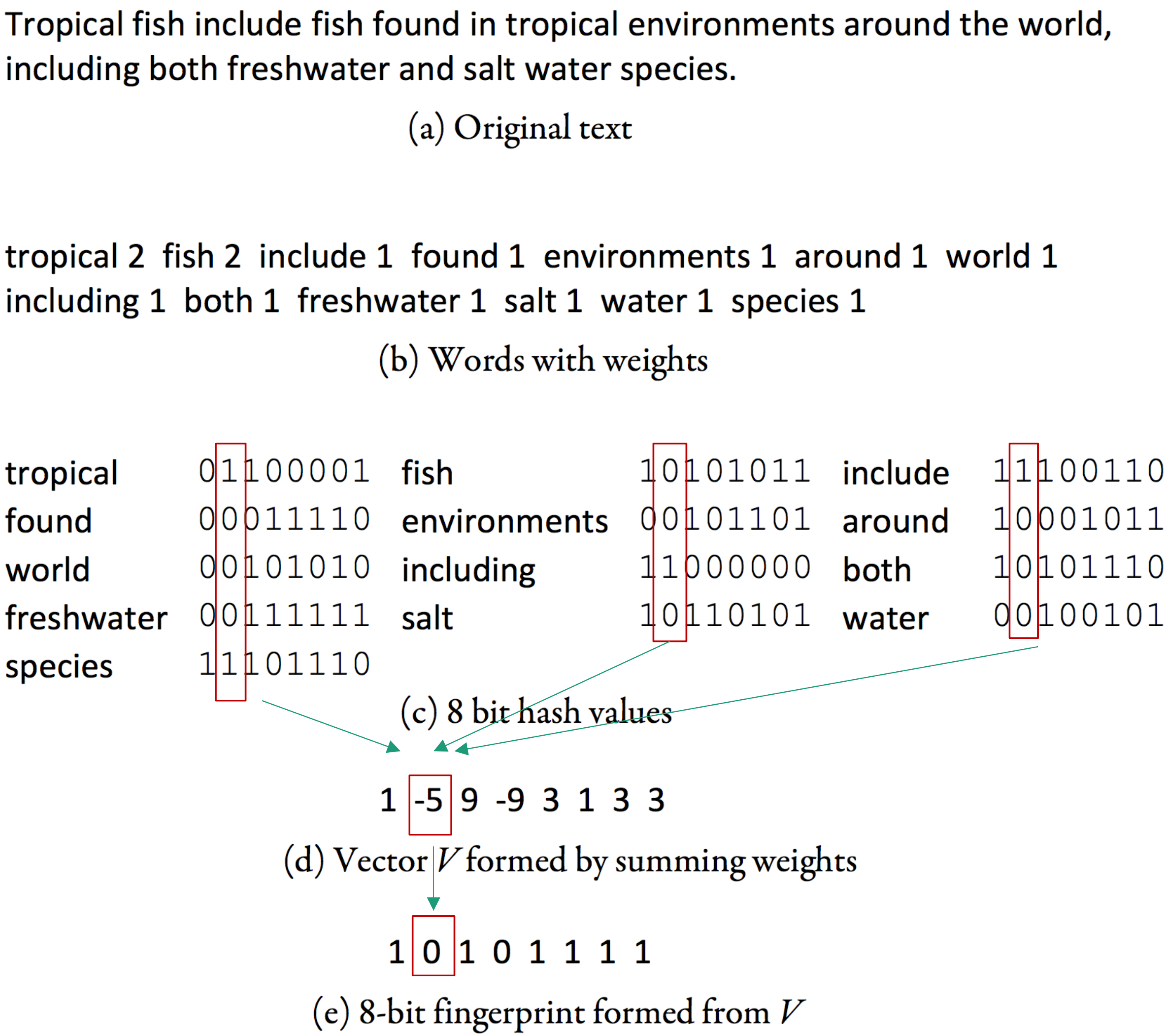
Simhash

- Documents are converted to set of features associated with weight
  - Features: word, shingle
  - Weight: frequency, TF-IDF
- Create *b* bit fingerprint of each document (e.g., b = 64)
- To compute similarity between two documents compute hamming distance of two documents



Algorithm 1 Simhash (u)

```
$W \leftarrow$ array of $b$ zeros
for $i \in \mathcal{F}(u)$ do                          ▷ Examine each feature
    $\phi_i \leftarrow$ UniformHash$(i)$               ▷ Compute $b$-bit hash
    for $j = 1$ to $b$ do                              ▷ Iterate through each bit
        if $\phi_{ij} = 1$ then                        ▷ $j$-th bit of $\phi_i$
            $W[j] \leftarrow W[j] + w_i$               ▷ Add feature weight
        else
            $W[j] \leftarrow W[j] - w_i$               ▷ Subtract feature weight
        end if
    end for
end for
for $j = 1$ to $b$ do                                  ▷ Revisit all bits
    if $W[j] \geq 0$ then
        $B[j] \leftarrow 1$                            ▷ Positive weight, set bit to 1
    else
        $B[j] \leftarrow 0$                            ▷ Negative weight, set bit to 0
    end if
end for
return array $B[1 \ldots b]$                           ▷ simhash
```



Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

(a) Original text

tropical 2 fish 2 include 1 found 1 environments 1 around 1 world 1 including 1 both 1 freshwater 1 salt 1 water 1 species 1

(b) Words with weights

tropical	01100001	fish	10101011	include	11100110
found	00011110	environments	00101101	around	10001011
world	00101010	including	11000000	both	10101110
freshwater	00111111	salt	10110101	water	00100101
species	11101110				

(c) 8 bit hash values

1 -5 9 -9 3 1 3 3

(d) Vector V formed by summing weights

1 0 1 0 1 1 1 1

(e) 8-bit fingerprint formed from V

How it works

Given a collection of vectors in R^d, we define a LSH: We choose a random vector $\vec{r}$. Corresponding to this vector $\vec{r}$, we define a hash function $h_{\vec{r}}$:

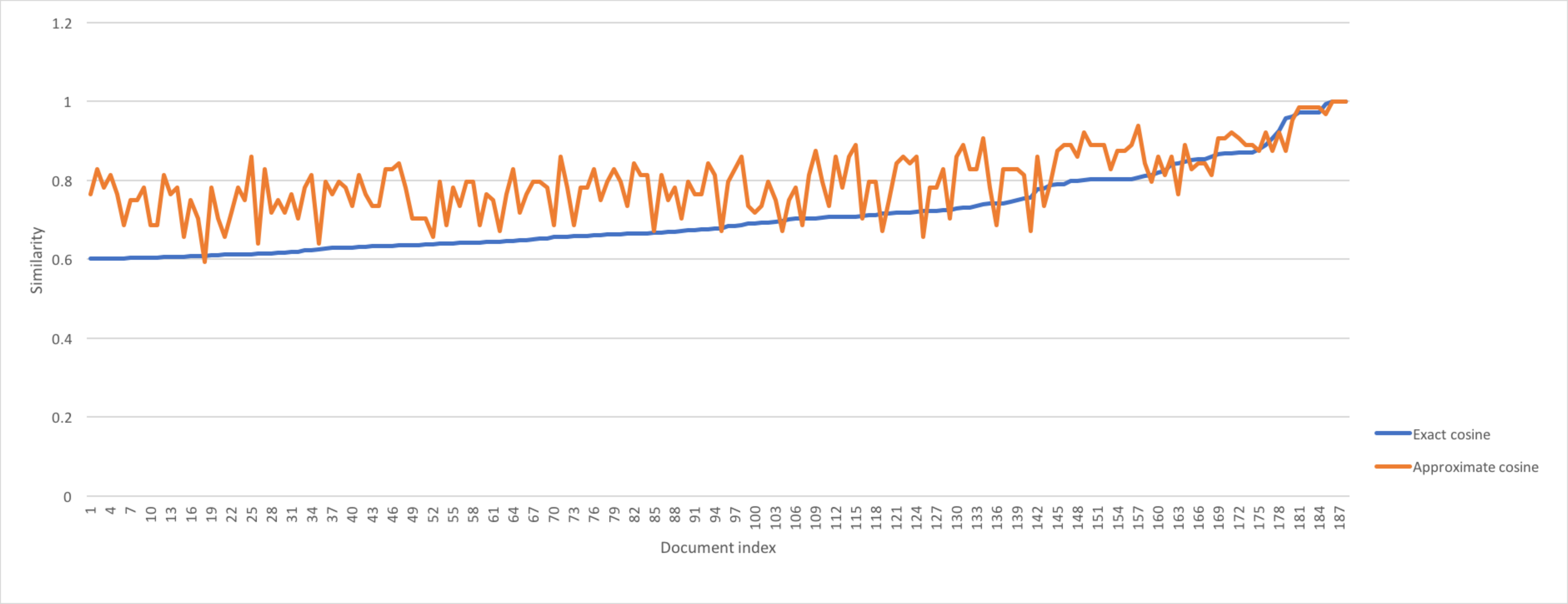
$$h_{\vec{r}} = \begin{cases} 1 & \text{if } \ \vec{r}.\vec{u} \geq 0 \\ 0 & \text{if } \ \vec{r}.\vec{u} < 0 \end{cases}$$

Then for vectors $\vec{u}$ and $\vec{v}$,

$$\mathbf{Pr}[h_{\vec{r}}(\vec{u}) = h_{\vec{r}}(\vec{v})] = 1 - \frac{\theta(\vec{u}, \vec{v})}{\pi}$$

$\theta(\vec{u}, \vec{v})$ refers to the angle between vectors $\vec{u}$ and $\vec{v}$. The function $1 - \frac{\theta(\vec{u}, \vec{v})}{\pi}$ is closely related to the function $1 - \cos(\theta)$. In fact it is always within a factor 0.878 from it.

Correlation



Hamming Distance Problem

- If *x* and *y* are two binary numbers of length *b*
  - Hamming distance $\mathrm{H}(x, y) = |\{i : x_i \neq \ y_i\}|$
    $= |\{i : x_i \oplus y_i = 1\}|$
  - Similarity $\mathrm{S}(x, y) = 1 - H(x, y)/b$
- If we have 10,000 documents. Then we need about 50 million comparisons.

Hamming Distance Problem

Index	Decimal	Hash(binary)	
1	37586	1001001011010010	
2	50086	1100001110100110	7
3	2648	0000101001011000	11
4	934	0000001110100110	9
5	40957	1001111111111101	9
6	2650	0000101001011010	9
7	64475	1111101111011011	7
8	40955	1001111111111011	4

Sort

Index	Decimal	Hash(binary)	
4	934	0000001110100110	
3	2648	0000101001011000	9
6	2650	0000101001011010	1
1	37586	1001001011010010	5
8	40955	1001111111111011	6
5	40957	1001111111111101	2
2	50086	1100001110100110	9
7	64475	1111101111011011	9

H(3, 6) = 1, H(8, 5) = 2 are adjacent
However, H(4, 2) = 2 which has fall apart.

Hamming Distance Problem

Rotate bits left twice →

Index	Decimal	Hash(binary)	
4	3736	0000111010011000	
3	10592	0010100101100000	9
6	10600	0010100101101000	1
1	19274	0100101101001010	5
8	32750	0111111111101110	6
5	32758	0111111111110110	2
2	3739	0000111010011011	9
7	61295	1110111101101111	9

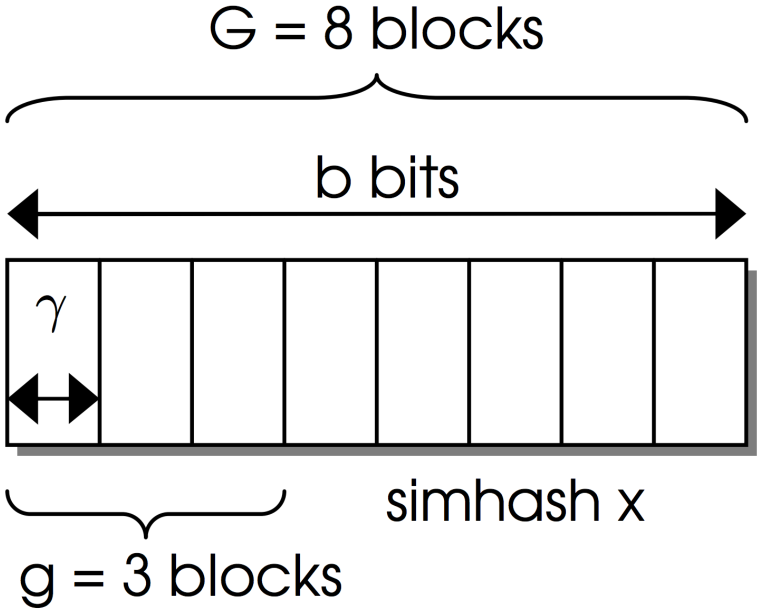
Sort →

Index	Decimal	Hash(binary)	
4	934	0000001110100110	
2	2648	0000111010011011	2
3	2650	0010100101100000	11
6	37586	0010100101101000	1
1	40955	0100101101001010	5
8	40957	0111111111101110	6
5	50086	0111111111110110	2
7	64475	1110111101101111	6

Now, H(4, 2) = 2 are adjacent

Block Permuted Hamming Search

- Let, h=3 and b=64
- Take G=8 blocks, so γ=64/8=8
- Select integer g between 1 and G-h, say g =3
- Comparison between x and y is guaranteed to have at least g exact block matches
- Use g blocks as header
- We can choose header in $\binom{G}{g}$ ways. Each works as a permutation π_i.



Conclusion

- Experiment on 1000 documents:
- Time taken by pairwise comparison: 625 sec
- Time taken by block permutation: 2.62 sec
- Number of pairs having similarity more than 90%: 19
- Detected pairs: 15

References

- Charikar, Moses S. "Similarity estimation techniques from rounding algorithms." In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pp. 380-388. ACM, 2002.
- Croft, W. Bruce, Donald Metzler, and Trevor Strohman. *Search engines: Information retrieval in practice*. Vol. 283. Reading: Addison-Wesley, 2010.
- Manku, Gurmeet Singh, Arvind Jain, and Anish Das Sarma. "Detecting near-duplicates for web crawling." In *Proceedings of the 16th international conference on World Wide Web*, pp. 141-150. ACM, 2007.
- Sood, Sadhan, and Dmitri Loguinov. "Probabilistic near-duplicate detection using simhash." In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pp. 1117-1126. ACM, 2011.
- The Simhash Algorithm. Web: http://matpalm.com/resemblance/simhash/