Probabilistic Near-Duplicate Detection Using Simhash

Sadhan Sood, Dmitri Loguinov

Presented by Matt Smith

Internet Research Lab
Department of Computer Science and Engineering
Texas A&M University

27 October 2011

Agenda

- Introduction
- Motivation and Objectives
- Simhash
- Bit Order
- Experiments and Results
- Conclusions and Future Work

Introduction

- Similarity matching is a common task in data mining; we are often interested in knowing which documents of a collection are "similar" to each other
- Usually involves representing documents by *d*-dimensional feature vectors and comparing those, but all-to-all comparison is infeasible for large collections
- Approximation algorithms such as *simhash*, trading some precision and recall for speed, are a promising technique for use on large collections

Introduction

- Simhash replaces a document's feature vector with a fixed-size fingerprint that preserves cosine similarity of the original vector space
- Main challenge: quickly find all pairs of fingerprints within a certain Hamming distance *h* of each other

Agenda

- Introduction
- Motivation and Objectives
- Simhash
- Bit Order
- Experiments and Results
- Conclusions and Future Work

Motivation

- A *feature vector* represents the subset of features present in a given document *u* of the collection, each feature being described by a real-valued weight
- Given typical values for average feature count and storage required per feature (e.g., 141 and 8 bytes respectively), all-to-all comparisons are completely infeasible
- A conversion to a fixed-size fingerprint of the feature vector (as done by Simhash) helps with storage and computational complexity concerns
  - Manku et al. [2007] showed that 64 bits is generally enough to capture similarity of much larger feature vectors

Motivation

- Even with the much faster Hamming distance calculation on this fingerprint, a sub-quadratic technique will be very desirable:
  - (Table is across all *n* pairs of crawled webpages)

n	Cosine $s(u,v)$		Hamming $H(x,y)$	
	Time	RAM	Time	RAM
1M	91 days	1.1 GB	34 min	8 MB
64M	1,020 years	70 GB	97 days	512 MB
8B	261K years	9 TB	68 years	64 GB

Objectives

- We consider two classes of matching problems
- Clustering: given one page, find *all* of its matches or near-duplicates
- Duplicate elimination: determine if there exists *at least* one match in the collection, without finding all matching documents
  - Can allow us to improve performance significantly by skipping the exhaustive search

Agenda

- Introduction
- Motivation and Objectives
- Simhash
- Bit Order
- Experiments and Results
- Conclusions and Future Work

Simhash

- The simhash algorithm operates as follows:
  - Initialize a vector *W* of weights to 0
  - Each feature *i* (word on a webpage, etc) is hashed with a uniformly random function
  - For each bit *j* of hash φ_i, add or subtract the feature weight w_i to/from W_j based on whether the bit is 0 or 1
- Example:

Feature	Hash	weight				
$word_1$	0101	0.05	-0.05	+0.05	-0.05	+0.05
$word_2$	1101	0.02	+0.02	+0.02	-0.02	+0.02
$word_3$	0001	0.01	-0.01	-0.01	-0.01	+0.01
$word_4$	1110	0.03	+0.03	+0.03	+0.03	-0.03
$word_5$	0100	0.05	-0.05	+0.05	-0.05	-0.05
$word_6$	0011	0.09	-0.09	-0.09	+0.09	+0.09
Σ *weight*			-0.15	+0.05	-0.01	+0.09
simhash			0	1	0	1

Simhash

- Next, we examine the issue of which bits are likely to differ between “similar” documents
  - Or, put another way, how likely it is for a given bit in the simhash to flip given minor changes to a document
  - Details of the model can be found in the paper; we just give an illustrative example here
- Main observation: examining the simhash weight vector, typically discarded, gives us insight into the bit-flipping question
- The bit with the smallest absolute weight value is the one most likely to be flipped by small changes to the document – called a “weak” bit

Simhash

- Consider making changes to the document represented in the previous table with simhash value 0101:

Feature	Hash	weight				
$word_1$	0101	0.05	-0.05	+0.05	-0.05	+0.05
$word_2$	1101	0.02	+0.02	+0.02	-0.02	+0.02
$word_3$	0001	0.01	-0.01	-0.01	-0.01	+0.01
$word_4$	1110	0.03	+0.03	+0.03	+0.03	-0.03
$word_5$	0100	0.05	-0.05	+0.05	-0.05	-0.05
$word_6$	0011	0.09	-0.09	-0.09	+0.09	+0.09
Σ *weight*			-0.15	+0.05	-0.01	+0.09
simhash			0	1	0	1

Simhash

- Removing two unimportant features (e.g., with the lowest weights):

Feature	Hash	weight				
$word_1$	0101	0.05	-0.05	+0.05	-0.05	+0.05
~~$word_2$~~	~~1101~~	~~0.02~~	~~+0.02~~	~~+0.02~~	~~-0.02~~	~~+0.02~~
~~$word_3$~~	~~0001~~	~~0.01~~	~~-0.01~~	~~-0.01~~	~~-0.01~~	~~+0.01~~
$word_4$	1110	0.03	+0.03	+0.03	+0.03	-0.03
$word_5$	0100	0.05	-0.05	+0.05	-0.05	-0.05
$word_6$	0011	0.09	-0.09	-0.09	+0.09	+0.09
Σ *weight*			-0.16	+0.04	+0.02	+0.06
simhash			0	1	**1**	1

single bit change

Simhash

- Removing two important features (e.g., with higher than average weights):
  - Note that bit 3 still flips, as last time

Feature	Hash	weight				
~~*word*$_1$~~	~~0101~~	~~0.05~~	~~-0.05~~	~~+0.05~~	~~-0.05~~	~~+0.05~~
word$_2$	1101	0.02	+0.02	+0.02	-0.02	+0.02
word$_3$	0001	0.01	-0.01	-0.01	-0.01	+0.01
word$_4$	1110	0.03	+0.03	+0.03	+0.03	-0.03
~~*word*$_5$~~	~~0100~~	~~0.05~~	~~-0.05~~	~~+0.05~~	~~-0.05~~	~~-0.05~~
word$_6$	0011	0.09	-0.09	-0.09	+0.09	+0.09
Σ *weight*			-0.05	-0.05	+0.09	+0.09
simhash			0	**0**	**1**	1

two-bit change

Agenda

- Introduction
- Motivation and Objectives
- Simhash
- Bit Order
- Experiments and Results
- Conclusions and Future Work

Bit Order

- If we only want to search to a Hamming distance h = 1; the problem is trivial
  - Simply generate a table with the simhash entries sorted by increasing absolute value of bit weight
- However, in practice we want a larger maximum distance – so how do we determine which is the optimal second bit to flip?
  - E.g., given an initial single bit flip with weight 0.01, do we next try the bit with -0.5, or the two-bit combination (-1.9, 0.01)?

Bit Order

- Here we sort the bits of document *u*'s hash according to the absolute value of their weight, and for convenience refer to "bit *i*" as the bit with the *i*-th lowest weight
- We then build a *Volatility Ordered Set Heap* (VOSH), which sorts bit combinations according to flip probability
  - Height of this heap corresponds to *b*, the # of hash bits
  - Details and algorithm are in the full paper, Section 5.1-5.2

Bit Order

- Main properties of this heap:
  - A parent node represents a better flip combination than its children; i.e., more likely to flip given small changes to *u*
  - Left child increments the last bit of the parent
  - Right child, if exists, increments the bit to the left of any gap in bit positions of the parent

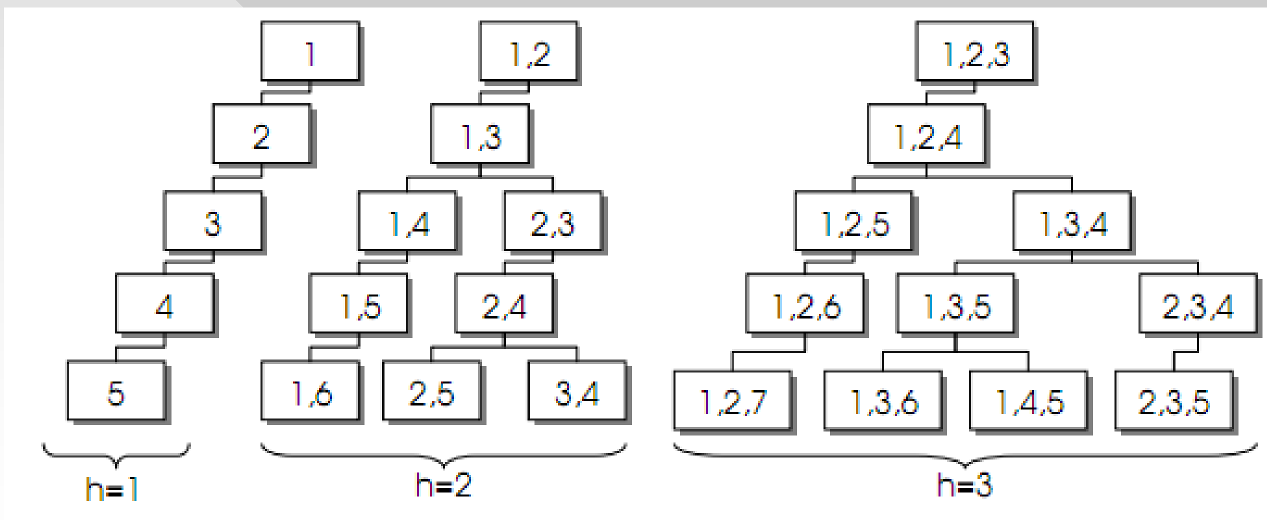


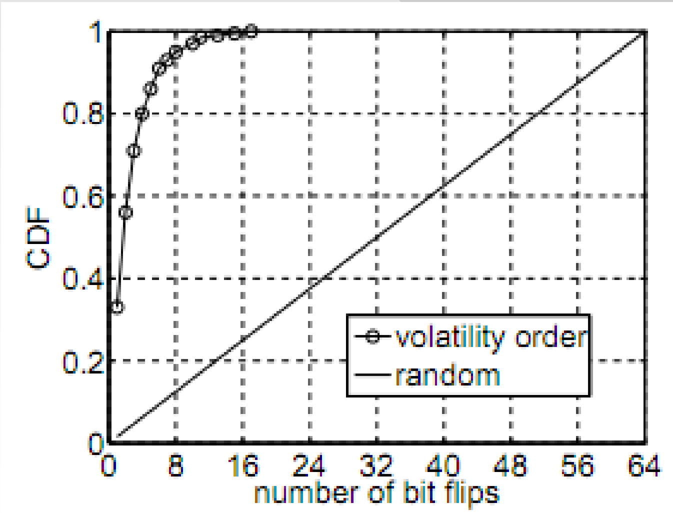
Figure 2: Top five levels of three volatility heaps.

Bit Order

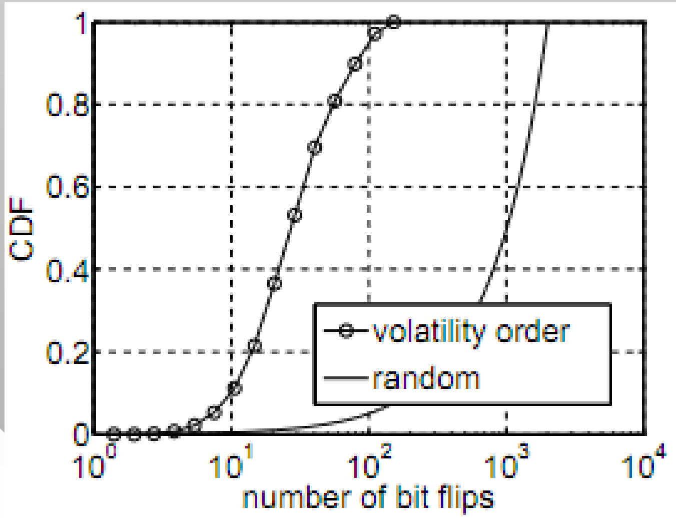
- We must decide at runtime which of the siblings at a given level in this heap is optimal, when we know the weight vector of the query simhash
- Additional max-heap is used to represent the "frontier" of yet-unexplored nodes
  - By calculating the expected change in value for flipping the bits represented in each node
  - At each step, the higher value node (i.e., the sibling that "lost" in the comparison) is placed, along with its children, in the max-heap

Bit Order

- Using *b* = 64 and *h* between 1 and 3, we examine the VOSH-based approach on 8M simhash pairs and compare it to randomly flipping bits
  - For *h* = 1, 30% of matches are found after only one flip; 80% after 4 flips, and 100% in 17 or fewer (vs. 64)
  - *h* = 2, 100% of matches found in 152 vs. 2016 flips



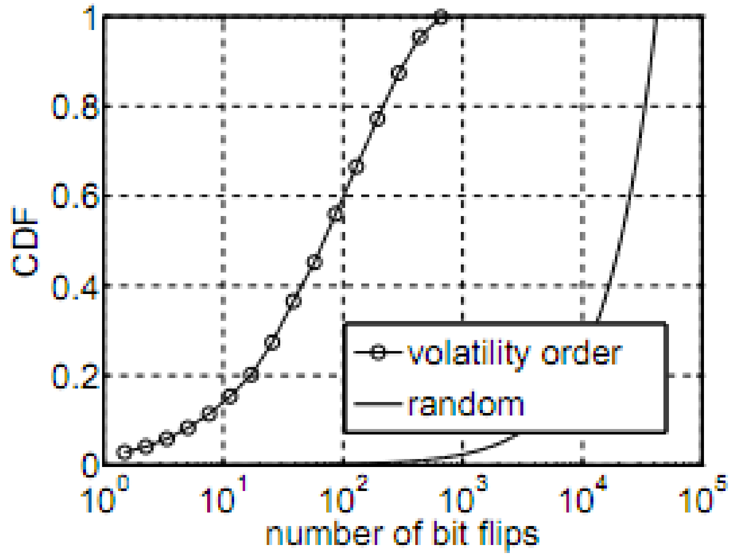
(a) $h = 1$



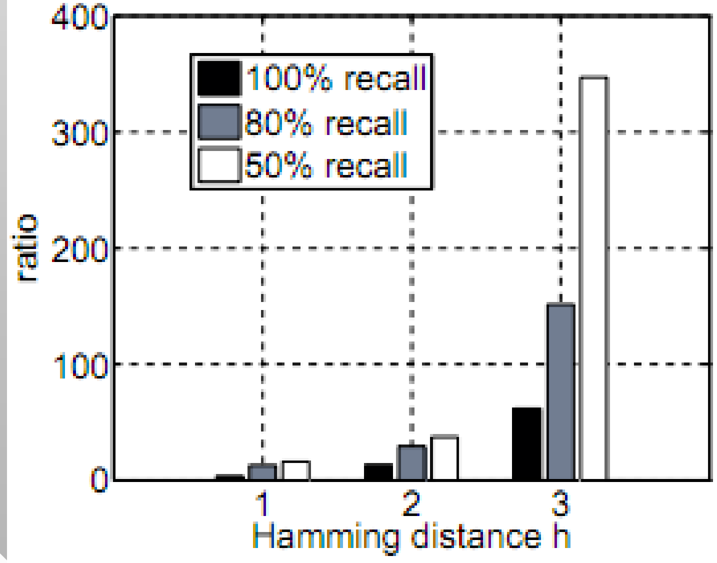
(b) $h = 2$

Bit Order

- Similar results for *h* = 3 (675 flips vs. 41,664)
- This difference increases with *h*, and as recall decreases



(c) $h = 3$



(d) speed-up ratio

Agenda

- Introduction
- Motivation and Objectives
- Simhash
- Bit Order
- Experiments and Results
- Conclusions and Future Work

Experiments and Results

- Dataset: 70M web pages from IRLbot web crawl (April 2008)
  - Feature weights calculated by normalized TF-IDF score of each word *i* on page *u*
  - Simhash fingerprint calculated with 64-bit MurmurHash function
- We compare our approach (PSM) to *Block Permuted Hamming Search* (BPHS), using the parameters suggested in the Manku paper
  - We normalize our RAM usage to BPHS' number of tables metric, see section 8.4 in the paper

Experiments and Results

Method	Tables	Time (sec)	Queries/sec
BPHS_A	4	65	154K
	10	53	189K
PSM_A	1.06	15.5	645K
	1.125	14.4	694K
	1.25	14.4	694K
	1.5	13.5	741K
	2	13.1	765K
BPHS_F	4	54	185K
	10	26	385K
PSM_F	1.06	6.9	1.4M
	1.125	6.7	1.5K
	1.25	6.4	1.56M
	1.5	6.26	1.6M
	2	6.25	1.6M

Table 2: Comparison of PSM to BPHS (online mode, 10M queries, 60M existing hashes).

Experiments and Results

- Scalability as dataset increases in size:

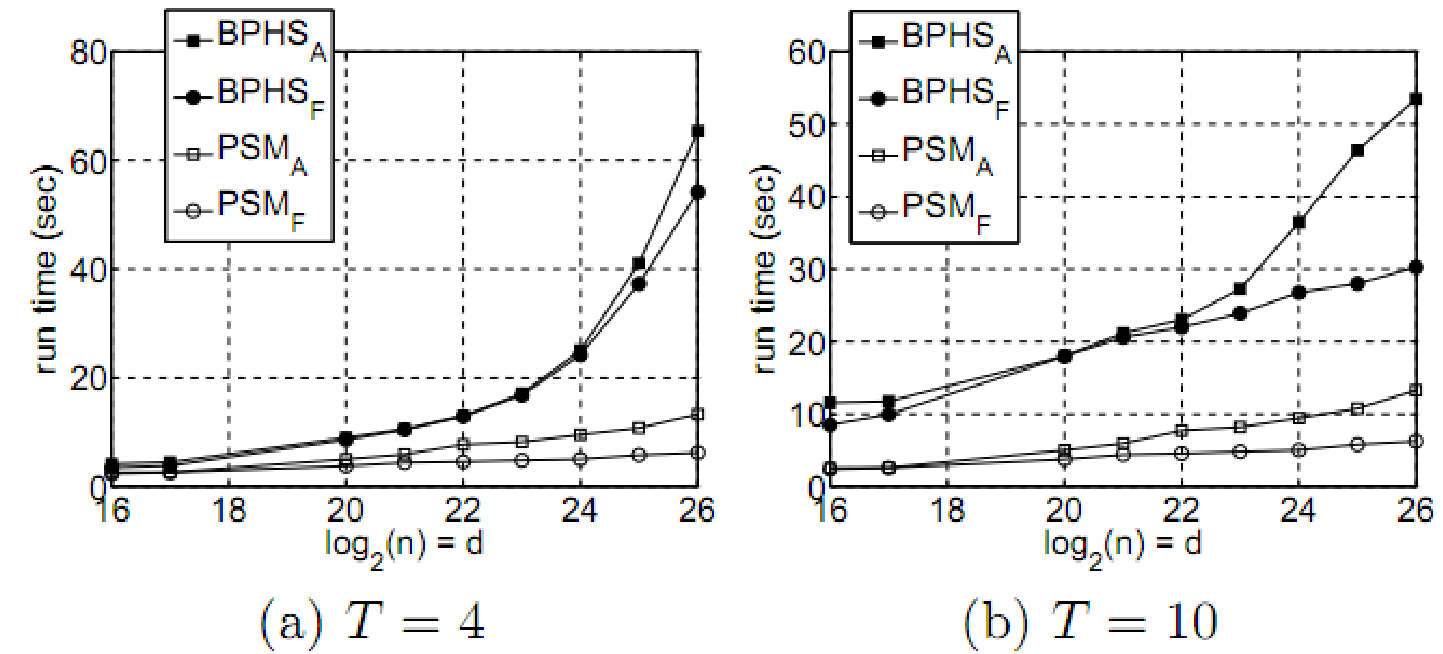
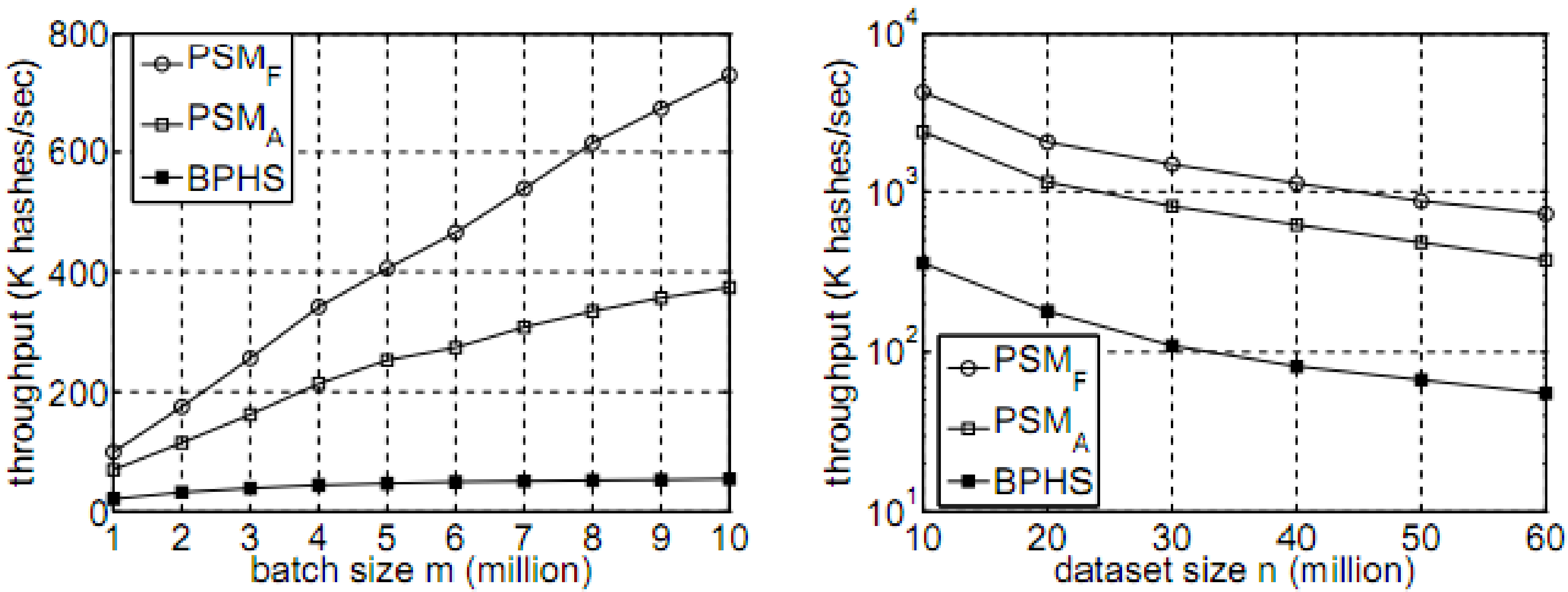


Figure 7: Scalability with dataset size (online mode, 10M queries, 60M existing hashes).

Experiments and Results

- Batch mode throughput
  - RAM usage is less important, but still smaller than BPHS



(a) speed vs m (b) speed vs n

Agenda

- Introduction
- Motivation and Objectives
- Simhash
- Bit Order
- Experiments and Results
- Conclusions and Future Work

Conclusions and Future Work

- By utilizing the weight vector usually discarded during simhash calculation, we can generate a model to predict which bits will be most likely to be flipped in near-duplicates
  - Result is a huge decrease in search time vs. exhaustive search, and the gap only widens if we're willing to sacrifice a little recall
- Future work involves analysis of feature selection techniques to improve clustering results, further overhead reduction