



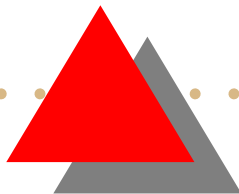
# Hashing Revisited

*Myths exposed, a new order revealed*

John Hamer

`J.Hamer@cs.auckland.ac.nz`

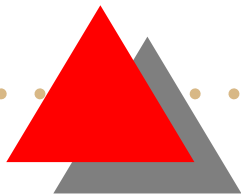
University of Auckland, New Zealand





# *Acknowledgements*

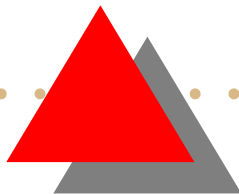
- The work presented here is almost entirely due to Robert Uzgalas (“Buz”), who single handedly reinvented hashing in the early 1990s.
- For reasons unknown, Robert never disseminated his work to the wide audience that it deserved. His paper, “Hashing Myths” remains unpublished, and his results are virtually unknown.
- It is time to make amends.





# *A brief history*

- 1953** Hans Peter Luhn writes an IBM memorandum setting out his ideas on “destroying every vestige of structure,” and defined what a general hash function should be.
- 1957** The first major paper on hashing, by Peterson, appears in the IBM Journal of R&D.
- 1973** Knuth writes “it is theoretically impossible to define a hash function that creates random data from the non-random data in actual files.”





# *Current Practice*

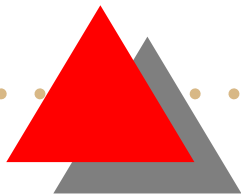
Textbook presentations of hashing are remarkably homogeneous. They invariably include

- a selection of hash functions (typically for hashing integers)
- an average-time analysis of chained hashing
- some disparaging comments on the worst-case scenario
- open addressing, covering collision handling through “double-hashing” and “linear probing”



# *Which is a shame, because*

- the hash functions studied are always ad-hoc, and perform poorly
- the practical worst-case requires an analysis of the longest expected chain length, but with a poor hash function there is no statistical theory on which to base such an analysis
- open addressing is never used in practice, and is in any event just an attempt at making use of the space available when using a poor hash function at extremely low load factors





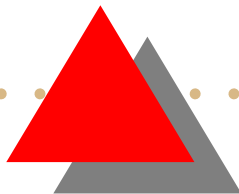
# *General hash functions*

Robert Uzgalas's contributions were

- a simple, general method for hashing bytes
- two simple, general hash combinators

The combinators allow excellent hash functions to be built for any data type.

The hash functions are so good they are virtually indistinguishable from a uniform random variable.





# Hashing a byte


To construct the hash function,

- Create an array, `buztable`, containing 256 randomly chosen integers. Done!

To hash the byte  $b$ ,

- return `buztable[b]`. Done!

Technical detail: for best results, the table should “unbiased”; i.e., each bit position when viewed as a column has an equal numbers of 1s and 0s.





# *Hash combinators*

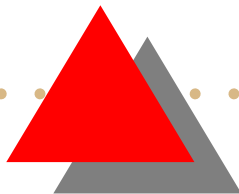
To hash an object that is larger than a byte, hash each of its component parts and combine the results together. Uzgalas identified two methods for combining a pair of hash values,  $a$  and  $b$ .

- If the order of the components matters, use

$$\text{rotate}(a) \text{ xor } b$$

- If the order does not matter, use

$$a \text{ xor } b$$





# *An example*

To hash a 16-bit integer

- break the integer into constituent bytes
- hash each byte
- combine the results with a rotating xor

In Java,

```
int hash( short x ) {  
    int a = buztable[ x >> 8 ];  
    int b = buztable[ x & 0xFF ];  
    return rotate(a) ^ b;  
}
```

# *Another example*

```
class Track {
    Album    album;
    String   name;
    int      duration;

    public int hashCode( ) {
        int h = album.hashCode( );
        h = rotate(h) ^ name.hashCode( );
        h = rotate(h) ^ hash( duration );
        return h;
    }
}
```



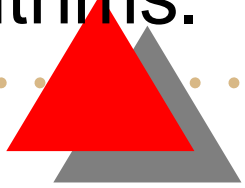
# Bring on the theory

Uzgalas's hash functions provide a bridge between hashing and statistical theory.

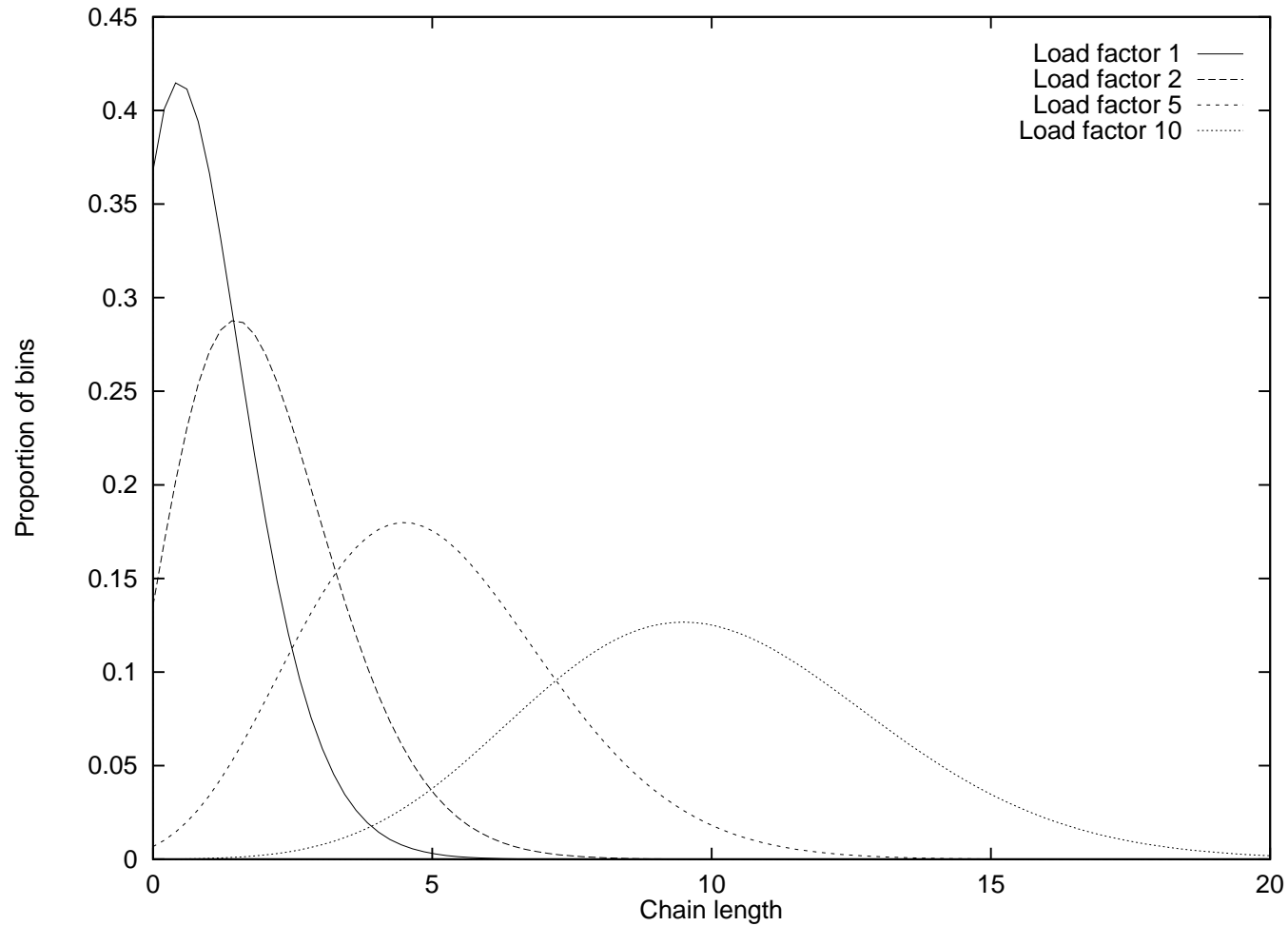
Statistics says that if you roll a fair  $n$ -sided dice  $m$  times, then the proportion of numbers that come up  $k$  times will be

$$\left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{m-k} \binom{m}{k} \approx \frac{\alpha^k}{e^\alpha k!}$$

where  $\alpha = m/n$  and  $e$  is the base of the natural logarithms.



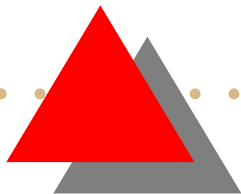
# Chain distribution by load





# *Worst case in practice*

The statistical theory of a uniform random variable allows the *real* worst case of hashing to be predicted accurately.



# A table of 1000 slots and 2000 elements

$k$	Prob. $k$ collisions
0	13%
1	27%
2	27%
3	18%
4	9%
5	3%
6	1%
7	0.3%

For a hash table with 1000 slots and 2000 elements (load factor 2) the theory predicts 1% (or about 10) of the slots to hold 6 elements, and 0.3% (or about 3) to hold 7 elements.

# Teaching ideas

- Download 1000 random integers in the range  $0-n$  from, say, `www.random.org`
- Calculate the distribution and compare to the theory
- Hash a variety of sources through a “buzhash” function (e.g., Java identifiers, numeric strings, consecutive bitstrings)

# Worthy diversions

- The exact formula for the distribution is an interesting one

$$\left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{m-k} \binom{m}{k}$$

- Coding  $\binom{m}{k}$  using the closed formula will generate some very large intermediate results for relatively small  $n, r$ .

$$\binom{n}{r} = \frac{n!}{r!(n-r)!}$$

# Motivating an inductive definition

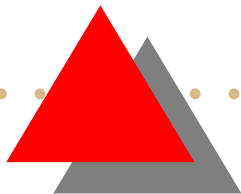
- The problem with managing large intermediate values is a great motivation for an inductive formula

$$\binom{n}{r} = \begin{cases} 1 & \text{if } r = 0 \text{ or } n = 1 \text{ or } n = r \\ n & \text{if } r = 1 \\ \binom{n}{r-1} + \binom{n-1}{r-1} & \text{otherwise} \end{cases}$$



# *Rounding, in a clash of extremes*

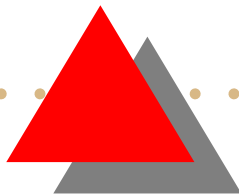
- Once  $\binom{n}{k}$  has been implemented (either as a recursive function or using bignums) a classic rounding problem appears
- The exact formula is a clash of extremes, multiplying a huge quantity by a vanishingly small one
- This creates a useful opening into the properties of floating point numbers





# Visualisations

- Plot points  $(h \bmod W, h \div W)$  for a range of hash values on a graphics frame. Vary the hash function and input source. Compare to a random source.
- Plot hash values as colours.
- Each of these provides a striking visual comparison between hash functions and the uniform random benchmark.





# *Closing comments*

- General hash functions completely change the study of hashing.
  - By closely simulating a uniform random variable, a rigorous theoretical treatment is possible.
  - Uzgaldas's method allows efficient general hash functions to be built with ease.
  - The topic provides scope for diversions into experimental validation, visualisation, overflow, induction, floating point arithmetic. . . all within an eminently practical setting.
- 