RATIONAL BLOOM FILTERS

Submitted by:

ROSS HEANEY

Supervisor:

DR MARC VAN DONGEN

Second Reader:

DR WHO

MSc Computing Science

School of Computer Science & Information Technology
University College, Cork

March 26, 2023

Ross Heaney: *Rational Bloom Filters,* MSc Computing Science, © March 26, 2023

Abstract

Bloom Filters are a type of space-efficient probabilistic data structure that can be used to test whether an element is a member of a set. That is when the Bloom Filter is queried 'does this element exist in the set?' and the Bloom Filter returns 'yes' when in fact the element does not exist in the set. We accept these false positives as by agreeing to occasionally get a false positive, we gain an enormous space advantage versus had we refused to accept a false positive. Using mathematics we can know in advance our false positive rate and tune the bloom filter accordingly. Namely, by setting the size of our bit array to $\frac{K}{S}\ln(2)$ we can reduce the false positive rate of the bloom filter to 2^{-K}, which is optimal. Given a maximum, allowed false positive rate, r, finding the optimal (minimal) value for k is easy. By assumption k is integral, which makes it difficult to tune a Bloom filter when the desired false positive rate is almost of the form 2^{-k}. For example, a small change in r may result in a significant increase in the size of the filter, especially when k is small. This thesis is about relaxing this integrality assumption and comparing to the state of the art.

Declaration

I confirm that, except where indicated through the proper use of citations and references, this is my original work and that I have not submitted it for any other course or degree.

Signed: _____

Ross Heaney
March 26, 2023

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Bloom Filters are a type of space-efficient probabilistic data structure that can be used to test whether an element is a member of a set. They have exploded in popularity as their usefulness is proportional to the size of the membership set. That is, the larger the size of the dataset we wish to query, the more useful Bloom Filters are. With ever-increasing amounts of data being produced and pipelined year after year, the more Bloom Filters have become almost necessary. In this first chapter I will discuss my thesis mission and the literature review plan. I then follow on with the background and related work in chapter 2. Specifically I discuss the origins of the Bloom Filter via the original paper by Bloom[1] and discuss important mathematical results by Kirsch and Mitzenmacher as well as a small but important detail from the book by Mitzenmacher and Upfal[3][4].

1.1 Motivations

As mentioned in the Introduction, bloom filters are a type space efficient probabilistic data structure. First conceived in the 1970s, Bloom Filters have seen widespread use ever since. Today they are used in a variety of applications, including network security, distributed systems, and databases. As data gets larger and larger it becomes more and more difficult to store and process it. Bloom Filters are a way to store and process data in a space efficient manner. This is done by using the so called 'Allowable Error Hashing' technique proposed in the landmark paper, by Burton Howard Bloom, in 1970[1]. This technique allows for a trade-off between the space efficiency and the accuracy of the data structure. More importantly, we can tune and optimize the Bloom Filter in relation to the accuracy and space efficiency of the data structure. Specifically this project and thesis titled 'Rational Bloom Filters' will focus on looking at new ways to tune and optimize the Bloom Filter with a specific focus on relaxing the integrality assumption of the number of hash functions required for the Bloom Filter to function properly.

1.2 Thesis Mission

Bloom Filters are based on hashing. Specifically hashing where we accept there will be some hash collisions that will give rise to so called 'false positives'. That is when the Bloom Filter is queried 'does this element exist in the set?' and the Bloom Filter returns 'yes' when in fact the element does not exist in the set. We accept these false positives as by agreeing to occasionally get a false positive, we gain an enormous space advantage versus had we refused to accept a false positive. Using mathematics we can know in advance our false positive rate and tune the bloom filter accordingly. Namely, by setting the size of our bit array to $\frac{K}{S}\ln(2)$ we can reduce the false positive rate of the bloom filter to 2^{-K}, which is optimal. Given a maximum, allowed false positive rate, r, finding the optimal (minimal) value for k is easy. By assumption k is integral, which makes it difficult to tune a Bloom filter when the desired false positive rate is almost of the form 2^{-k}. For example, a small change in r may result in a significant increase in the size of the filter, especially when k is small. This thesis is about relaxing this integrality assumption and comparing to the state of the art.

Chapter 2

Background and Literature Review

In this chapter I will discuss the background and related work in the area of bloom filters. Specifically I will discuss the concept of a bloom filter before discussing the original paper by Bloom and the recent work done on bloom filters as it relates to this thesis. Throughout, I will be discussing important mathematical results that are relevant to the thesis.

2.1 Concept of a Bloom Filter

The concept of a bloom filter is summarized nicely in this diagram 2.1. The core idea behind bloom filters is the concept of hashing. Hashing is simply the transformation of an input value to an output value. In a bloom filter, sometimes a different input will transform to the same output. This is called a hash collision. In certain use cases such as cryptography, hash collisons can be very dangerous, but in the case of Bloom Filters, we accept hash collisions. So, naturally the question is why do we allow hash collisons in Bloom Filters? The answer to that question is space efficiency. By allowing hash collisions we can reduce the space required to store the data structure. This will be discussed in more detail in the next section. For now, it is important to note that the core idea behind a bloom filter is the concept of hashing and allowing hash collisions. The core use case of a Bloom Filter is testing set membership. This is where we can query whether 'x' is present in the Bloom Filter or not. The Bloom Filter will respond either "possibly in the set" or "definitely not in the set". The Bloom Filter will never return a *false negative* but *false positives* are possible.

2.2 Original Paper by Burton Howard Bloom

The paper was the first paper to introduce the idea of *Allowable Error Hashing*, which is the basis of Bloom Filters. The essence of Bloom Filters is that set membership, is this item in this set?, can be answered in one of two ways. The first way is to return

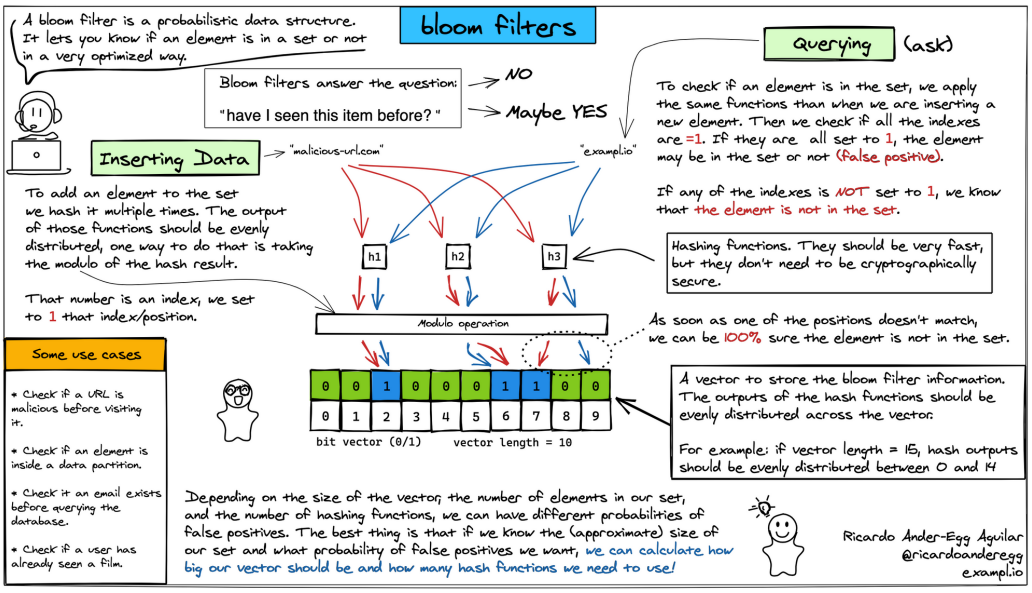

Figure 2.1: Bloom Filter Summary

yes this item is in the set and the other way is to say no this item is not in the set. Bloom Filters **guarantee** that false negatives are not possible, while occasionally a false positive is possible. In other words when a Bloom Filter is queried whether an item is in the set or not, it will return "possibly in the set" or "definitely not in the set". The underlying mechanism for the construction of a Bloom Filter is a number of hash functions that map elements of a set to indices on an array. This can be seen in Figure 2.1. The paper introduced the key concept of an *Allowable Error Rate*. By accepting that we will get a false positive occasionally, we can achieve remarkable space efficiency. The paper's proposed idea is analysed quite nicely via a mathematical comparison between the conventional error-free hashing, state of the art in the 1970s, and the authors proposed *Allowable Error Hashing*. It should be noted that some errors have been found in the original paper and updates published[2]. The errors found are mostly to do with edge cases and do not necessarily diminish the original paper's contribution. The paper discusses an example of a hyphenation algorithm. Suppose we have a dictionary of 500,000 words of which 90% follow simple hyphenation rules but the remaining 10% require expensive disk access to retrieve very specific hyphenation rules that are stored on the disk. By applying the authors *Allowable Error Hashing* technique, disk access is greatly reduced. A hash area only 15% of the total size needed by an error-free hash cuts down on 85% of the disk access required. The main idea is not that error-free hashing is bad but rather there is a proposed alternative called *Allowable Error Hashing* that is better suited to certain applications.

2.3 Less Hashing, Same Performance

Bibliography

[1] Burton H Bloom. "Space/time trade-offs in hash coding with allowable errors". In: *Communications of the ACM* 13.7 [1970], pp. 422–426.

[2] Prosenjit Bose et al. "On the false-positive rate of Bloom filters". In: *Information Processing Letters* 108.4 [2008], pp. 210–213.

[3] Adam Kirsch, and Michael Mitzenmacher. "Less hashing, same performance: Building a better bloom filter". In: *Algorithms–ESA 2006: 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006. Proceedings 14*. Springer. 2006, pp. 456–467.

[4] Michael Mitzenmacher, and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.