Bloom Filter in Bioinformatics

Aleksey Kladov

aleksey.kladov@gmail.com

February 15, 2013

Implementations

- Hash table
- Search tree
- Bloom filter

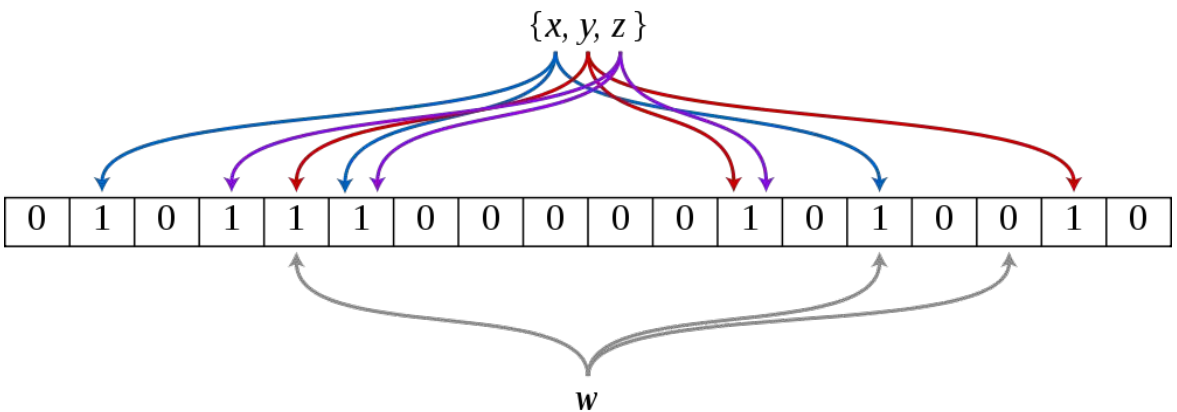
Why Bloom is awesome?

- Asymptotically **not** faster then a hash table
- No delete operation
- It simply doesn't work!

Meet the Bloom!

- Actually, it's all from Wikipedia :)
- Burton Howard Bloom, 1970
- m-bit vector B
- k hash functions: $element \longrightarrow [0, m)$

- Want to add element e?
- Compute $h_1(e), h_2(e), \ldots, h_k(e)$
- Compute $H(e)$ m-bit vetcor with k ones at $h_1(e), h_2(e), \ldots, h_k(e)$
- Let $B = B \vee H(e)$



- Want to check if element e was inserted in B?
- Check if $B \wedge H(e) == H(e)$
- If yes then e **may be** in the set
- If no then e is definitely not in the set
- False positives =(

From Wikipedia

A Bloom filter with 1% error and an optimal value of k, requires only about 9.6 bits per element — regardless of the size of the elements.

k, m, n and other letters

- k – number of hash functions
- m – length of the vector
- p – false positive rate
- n – number of elements in the set

Probability that i^{th} bit is **zero** after one insertion	$(1 - \frac{1}{m})^k$
After n insertions	$(1 - \frac{1}{m})^{kn}$
Probability that i^{th} bit is **one** after n insertions	$1 - (1 - \frac{1}{m})^{kn}$
Probability of positive answer	$(1 - (1 - \frac{1}{m})^{kn})^k$

Recall that $(1 - \frac{1}{m})^m \approx e^{-1}$, plug it into the last formula and we get **false positive rate estimate**

$$\approx (1 - e^{\frac{-kn}{m}})^k$$

Can we choose k to minimize false positive rate?

$$\min_k (1 - e^{\frac{-kn}{m}})^k$$

Optimal k

$$\frac{m}{n} \ln 2$$

False positive rate

$$2^{-\frac{m}{n} \ln 2}$$

Number of bits per item for a given p

$$-\frac{\ln p}{(\ln 2)^2}$$


Excellent

- Naive: hash table requires a lot of memory =(
- $\approx$ half of the k-mers are singleton!
- Increase in coverege leads to more singletons (linear dependency)
- Most applications need solid k-mers

Bloom filter!

Paper "Efficient counting of k-mers in DNA sequences using a bloom filter" by Páll Melsted and Jonathan K Pritchard, 2011

- Create a bloom filter and a hash table
- If k-mer is not in filter – insert into filter
- If k-mer is in filter – insert into table with count 2 or update count.

Resulting counts are higher then true counts at most by one.
To find precise counts make a second pass.
2x memory savings, assuming if half of the k-mers are unique

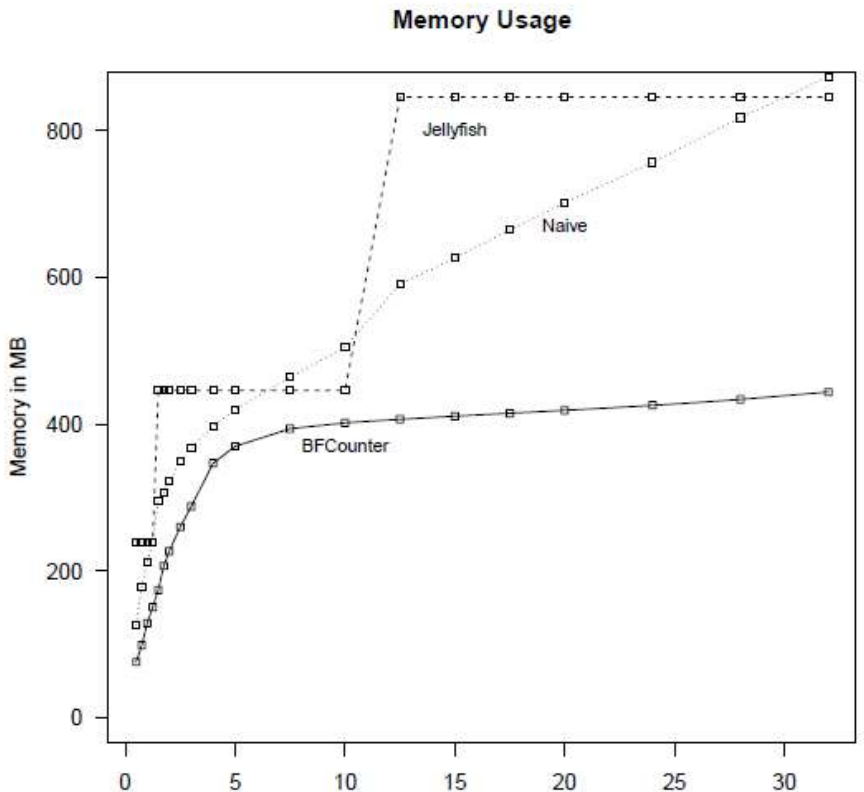


Figure 1 : Memory usage for BFCounter, Jellyfish and Google Dense at different coverage levels(Chromosome 21 data)

Speeding it up I

Paper "Exact Pattern Matching with Feed-Forward Bloom Filters" by Iulian Moraru and David G. Andersen, 2011

Problem

Given the huge text and lots of patterns of the same length, find all occurences of patterns in the text.

Solution

- Create P and T Bloom filters
- Insert all patterns in P
- Search substring of text in filter
- For each substring of text that is in P, remember it's positon and add it to the T
- Filter out all patterns that are not in T
- Check all positions against reduced set of patterns

Poor cache behavior

Split filter into small cache part and large memory part

Lot's of hash functions

- Fast rolling hash functions
- Hash functions don't need to be independent: linear combinations are OK

This is the end

\end {document}