Optimization of bloom filter using simulated annealing for spam filtering

Arulanand Natarajan¹, Subramanian S² and Premalatha K³
¹Anna University of Technology, Coimbatore, TN, India
²Sri Krishna College of Engineering and Technology Coimbatore, TN, India.
³Bannari Amman Institute of Technology, Erode, TN, India.

ARTICLE INFO

Article history:
Received: 21 September 2011;
Received in revised form: 29 November 2011;
Accepted: 17 December 2011;

Keywords
Bin Bloom Filter,
Bloom Filter,
Spam word,
Hash function,
Simulated Annealing,
False positive rate.

Introduction

A spam filter is a program that is used to detect unsolicited and unwanted email and prevent those messages from getting into user's inbox. A spam filter looks for certain criteria on which it stands decisions. For example, it can be set to look for particular words in the subject line of messages and to exclude these from the user's inbox. This method is not effective, because often it is omitting perfectly legitimate messages and letting actual spam through. The strategies used to block spam are diverse and includes many promising techniques. Some of the strategies like black list filter, white list / verification filters rule based ranking and naïve bayesian filtering are used to identify the spam.

A Bloom filter presents a very attractive option for string matching (Bloom 1970). It is a space efficient randomized data structure that stores a set of signatures efficiently by computing multiple hash functions on each member of the set. It queries a database of strings to verify for the membership of a particular string. The answer to this query can be a false positive but never be a false negative. The computation time required for performing the query is independent of the number of signatures in the database and the amount of memory required by a Bloom filter for each signature is independent of its length (Feng et al 2002).

This paper presents a BBF which allocates different false positive rates to different strings depending on the significance of spam words and gives a solution to make the total membership invalidation cost minimum. BBF groups strings into different bins via smoothing by bin means technique. The number of strings to be grouped and false positive rate of each bin is identified through SA which minimizes the total membership invalidation cost. This paper examines different number of bins for given set of strings, their false positive rates and number of strings in every bin to minimize the total membership invalidation cost.

The organization of this paper is as follows. Section 2 deals with the standard BF. Section 3 presents the SA technique. Section 4 explains the optimized BBF using SA. Performance evaluation of the BBF with standard BF is discussed in section 5.

Bloom Filter

Bloom filters (Bloom 1970) are compact data structures for probabilistic representation of a set in order to support membership queries. This compact representation is the payoff for allowing a small rate of false positives in membership queries which might incorrectly recognize an element as member of the set.

Given a string S the BF computes k hash functions on it producing k hash values and sets k bits in an m-bit long vector at the addresses corresponding to the k hash values. The value of k ranges from 1 to m. The same procedure is repeated for all the members of the set. This process is called programming of the filter. The query process is similar to programming, where a string whose membership is to be verified is input to the filter. The bits in the m-bit long vector at the locations corresponding to the k hash values are looked up. If at least one of these k bits is not found in the set then the string is declared to be a nonmember of the set. If all the bits are found to be set then the string is said to belong to the set with a certain probability. This uncertainty in the membership comes from the fact that those k bits in the m-bit vector can be set by any other n-1 members. Thus finding a bit set does not necessarily imply that it was set by the particular string being queried. However, finding a bit not set certainly implies that the string does not belong to the set.

In order to store a given element into the bit array, each hash function must be applied to it and, based on the return
value \( r \) of each function \( r_1, r_2, \ldots, r_k \), the bit with the offset \( r \) is set to 1. Since there are \( k \) hash functions, up to \( k \) bits in the bit array are set to 1 (it might be less because several hash functions might return the same value). Figure 1 is an example where \( m=16, k=4 \) and \( e \) is the element to be stored in the bit array.

![Figure 1 Bloom Filter](image)

One important feature of BF is that there is a clear tradeoff between the size of the filter and the rate of false positives. The false positive rate of BF is

\[
f = \left(1 - e^{-\frac{kn}{m}}\right)^k = \exp(k \ln(1 - e^{-\frac{kn}{m}}))
\]

(1)

Let \( g = k \ln(1 - e^{-\frac{kn}{m}}) \). Minimizing the false positive probability \( f \) is equivalent to minimizing with respect to \( k \).

\[
\frac{df}{dk} = \ln \left(1 - e^{-\frac{kn}{m}}\right) + \frac{kn}{m} e^{-\frac{kn}{m}} (2 - \frac{km}{m})\]

(2)

The derivative equals 0 when \( k_{\text{min}} = (\ln 2)(m/n) \). In this case the false positive probability \( f \) is:

\[
f(k_{\text{min}}) = (1 - p)^{k_{\text{min}}} = \left(\frac{1}{2}\right)^{\ln 2} = (0.618)^{\ln 2}
\]

(3)

of course \( k \) should be an integer, so \( k \) is \( \lceil \ln 2 \cdot (m/n) \rceil \)

The BF has been widely used in many database applications (Mullin 1990; Mackert and Lohman, 1986). It is applied in networking literature (Broder and Mitzenmacher, 2005).

A BF can be used as a summarizing technique to aid global collaboration in peer-to-peer networks (Kubiatowicz et al., 2000; Li et al., 2002; Cuena-Acuna et al., 2003). It supports probabilistic algorithms for routing and locating resources (Rhee and Kubiatowicz 2004; Hodes et al., 2002; Reynolds and Vahdat, 2003; Bauer et al., 2004) and share Web cache information (Fan et al., 2000). BFs have great potential for representing a set in main memory (Peter and Panagiotis, 2004) in stand-alone applications.

BFs have been used to provide a probabilistic approach for explicit state model checking of finite-state transition systems (Peter and Panagiotis, 2004). It is used to summarize the contents of stream data in memory (Jin et al., 2004; Deng and Rafiei, 2006), to store the states of flows in the on-chip memory at networking devices (Bonomi et al., 2006), and to store the statistical values of tokens to speed up the statistical-based Bayesian filters (Li and Zhong, 2006). The variations of BFs are compressed Bloom filters (Mitzenmacher, 2002), counting Bloom filters (Fan et al., 2000), distance-sensitive Bloom filters (Kirsch and Mitzenmacher, 2006), Bloom filters with two hash functions (Kirsch and Mitzenmacher, 2006), spacecode Bloom filters (Kumar et al., 2004), spectral Bloom filters (Cohen and Matias, 2003), generalized Bloom filters (Laufer et al., 2005), Bloomier filters (Chazelle et al., 2004), and Bloom filters based on partitioned hashing (Hao et al., 2007).

**Simulated annealing**

In an optimization problem, often the solution space has many local minima. A simple local search algorithm proceeds by choosing random initial solution and generating a neighbor from that solution. If it is a minimum fitness transition then the neighboring solution is accepted. Such an algorithm has the drawback of often converging to a local minimum. The SA (Dowsland 1995) avoids getting trapped in a local minimum by accepting cost increasing neighbors with some probability. It solves this problem by allowing worse moves (lesser quality) to be taken some of the time. That is, it allows some uphill steps so that it can escape from local minima. In SA, first an initial solution is randomly generated, and a neighbor is found and is accepted with a probability of min \( (1, \exp^{-\Delta E/T}) \), where \( \Delta E \) is the cost difference and \( T \) is the control parameter corresponding to the temperature of the physical analogy and will be called temperature. On slow reduction of temperature, the algorithm converges to the global minimum. Among its advantages are the relative ease of implementation and the ability to provide reasonably good solutions for many combinatorial problems. Simulated Annealing is inherently sequential and hence very slow for problems with large search spaces.

**Algorithm**

Set \( X \) a initial configuration

Set \( E \) as Eval(\( X \))

Set \( T \) as high temperature and frozen is false while (!frozen)

repeat

Choose a random move \( i \) from the move set

Set \( E \) as Eval(move(\( X, i \))

if \( E < E_i \) then

set \( X \) as move(\( X, i \))

set \( E \) as \( E \)

else accept the move with probability \( \exp^{-\Delta E/T} \) even though things get worse until the stem is in thermal equilibrium at \( T \)

if \( (E \) is still decreasing over the last few temperatures) reduce \( T \)

else

assign frozen is true

**Bloom Filter Optimization using SA**

**Bin bloom filter**

A BBF is a data structure considering weight for spam word. It groups spam words into different bins depending on their weight. It incorporates the information on the spam word weights and the membership likelihood of the spam words into its optimal design. In BBF a high cost bin lower false positive probability and a low cost bin has higher false positive probability. The false positive rate and number of strings to be stored is identified through optimization technique SA which minimize the total membership invalidation cost. Figure 3 shows Bin Bloom Filter with its tuple \( \langle n, f, w \rangle \) configuration.

![Figure 3 Bin Bloom Filter](image)
Problem Definition

Consider a standard supervised learning problem with a set of training data \( D = \{<Y_i, Z_i>, ..., <Y_i, Z_i>, ..., <Y_i, Z_i>\} \), where \( Y_i \) is an instance represented as a single feature vector, \( Z_i = C(Y_i) \) is the target value of \( Y_i \), where \( C \) is the target function. Where \( Y_1, Y_2, ..., Y_i \) set of text document collection \( C \) is a class label to classify into spam or legitimate (non-spam). The collection is represented into feature vector by the text documents are converted to normalized case, and tokenized them, splitting on non-letters. The stop words are eliminated. The spam weights for words are calculated from the set. This weight value indicates its probable belongings to spam or legitimate. The weight values are discretized and assigned for different Bins. The tuple to describe the Bin Bloom Filter is, \( \{ \{ n_1, n_2, ..., n_k \}, \{ w_1, w_2, ..., w_k \}, m, \{ k_1, k_2, ..., k_l \}, \{ f_1, f_2, ..., f_l \} \} \). It is an optimization problem to find the value of \( n \) and \( f \) that to minimize the total membership invalidation cost. For membership testing the total cost of the set is the sum of the invalidation cost of each subset. The total membership invalidation cost (Xie et al., 2005) is given as:

\[
E = n_f w_1 + n_f w_2 + ... + n_f w_L
\]

The total membership invalidation cost or the Evaluation function is

\[
E(L) = \sum_{i=1}^{L} n_i w_i f_i
\]

where \( \sum_{i=1}^{L} n_i = N \)

\( N \)- Total number of strings in a spam set.

\[
\eta = \left(\frac{1}{2}\right) \ln 2 \left( \frac{m}{n} \sqrt{\frac{i}{f}} \right)
\]

\[
\eta = \ln (\eta) \quad (1 \leq i \leq L)
\]

Weight assignment

The first step for assigning weight to spam words is estimating the word probability that depends on word frequency. Word frequency is measured by the number of occurrences of a specific word in the document. Estimating probabilities is achieved using Bayes conditional probability theorem according to which the probability of a word given that the message is spam can be estimated as follows:

\[
P_s = \frac{f_s}{N_{s2}}
\]

\[
P_{as} = \frac{f_{as}}{N_{as}}
\]

\( P_s \) is the probability of a word given the mail is spam.

\( P_{as} \) is the probability of a word given the mail is legitimate.

\( f_s \) is the frequency of word in the spam documents.

\( f_{as} \) is frequency of words in the legitimate documents.

\( N_s \) is the total spam documents.

\( N_{as} \) is the total legitimate documents.

The next step is calculating word weights. Estimating a weight for each word is based on its frequency and its probability in spam mail documents and non-spam mail documents. The weight of every word is estimated using the formula:

\[
weight_{word} = \frac{x^L}{f_{nl}}
\]

This weight value is based on text collection containing spam messages and non-spam messages. The word weights are estimated for spam list during the training process and stored in a separate text document.

Initial solution

In the context of Bin bloom filter, a chromosome represents number of bloom filters with its number of words to be stored, false positive rate and its weight. That is, each chromosome \( X_i \) is constructed as follows:

\[
X = \{ n_1, f_1, w_1, n_2, f_2, w_2, ..., n_k, f_k, w_k \}
\]

Figure 4 Initial Solution

where \( n_i, f_i \) and \( w_i \) refer respectively the number of words, false positive rate and average weight of the jth bin. A set of 3 genes \( <n,f,w> \) encodes a protein – a trait, that is a single bin. The false positive rate \( f_i \) can be obtained from equation (6) where \( n_i \) is drawn from the particle dimension, \( m \) is known in advance and \( k \) is calculated from equation (8).

SA solves problem by allowing worse moves to be taken some of the time. That is, it allows some uphill steps so that it can escape from local minima.

The maximum cost function difference between one neighbour and another may be taken as a starting temperature. Another method, suggested in (Rayward-Smith, 1996), is to start with a very high temperature and cool it rapidly until about 60% of worst solutions are being accepted. This forms the real starting temperature and it can now be cooled more slowly. Dowsland (1995) suggested for heating the system until a certain proportion of worse solutions are accepted and then slow cooling can start.

The decrement of temperature is important to the success of SA. One way to decrement the temperature is a simple linear method. An alternative is a geometric decrement is \( T = T \alpha \) where \( \alpha < 1 \).

Experimental results

The total number of strings taken for testing is 3000 and their weights are ranging from 0.0005 to 5. The size of the BF is 1024. These experimental values are tested for bin size 10, 11 and 12. The Temperature T is assigned to BF size 1024 and T is decremented using the formula \( T = T \alpha \) where \( \alpha = 0.9 \).

Since Bloom Filter allows false positive, the membership invalidation cost is unavoidable. For BBF, the total membership invalidation cost is expressed in equation (4). In standard BF, different weights in different bins into consideration, the total membership invalidation cost is then as follows:

\[
F_{standard} = (n_1 w_1 + n_2 w_2 + ... + n_k w_k) / f
\]

\[
F_{standard} (L) = f \sum_{i=1}^{L} n_i w_i
\]

Figures 5,6 and 7 show the membership invalidation cost for standard BF with BBF using SA for the bin sizes 10,11 and 12 respectively.

Figure 5 Membership invalidation cost for bin size 10
the total membership invalidation cost remarkably standard BF respectively. The experimental results exhibit that average weight of the strings present in a bin; the fourth column, less value compared with the original standard BF.

The fifth and sixth column show the total membership invalidation cost of each bin in BBF and false positive rate of each bin. The experimental results exhibit that the total membership invalidation cost of the BBF remarkably less value compared with the original standard BF.

Conclusion

Bloom filters are simple randomized data structures that are useful in practice. The BBF is an extension of BF, and inherits the best feature of BF such as time and space saving. The BBF treats strings in a set in a different way depending on their significance, groups the strings into bins and allocates different false positive rate to different bins. Important spam words have lower false positive rate than less significant words. SA is an optimization technique which chooses a random move from the neighbourhood. If the move is better than its current position simulated annealing will always take it. If the move is worse then it will be accepted based on some probability. SA technique is applied in this paper to minimize the total membership invalidation cost. Experiment results show that the total membership invalidation cost in spam filtering with different false positive rate of BBF performs better than standard BF.

References


Table 1: Values obtained from SA at 100th iteration

<table>
<thead>
<tr>
<th>Bin size</th>
<th>Number of Strings</th>
<th>Average String weight</th>
<th>False positive rate</th>
<th>Cost of BBF</th>
<th>Cost of Standard BF</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>96, 129</td>
<td>4.906, 4.663</td>
<td>0.0059, 0.0221</td>
<td>6119.99</td>
<td>6340.83</td>
</tr>
<tr>
<td></td>
<td>174, 212</td>
<td>4.424, 4.101</td>
<td>0.0592, 0.0982</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>319, 321</td>
<td>3.694, 3.190</td>
<td>0.2139, 0.2159</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>383, 385</td>
<td>2.607, 1.966</td>
<td>0.2767, 0.2786</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>490, 491</td>
<td>1.250, 0.385</td>
<td>0.3664, 0.3671</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>45, 45</td>
<td>4.956, 4.757</td>
<td>1.79E-05, 1.79E-05</td>
<td>6122.02</td>
<td>6336.61</td>
</tr>
<tr>
<td></td>
<td>124, 169</td>
<td>4.677, 4.445</td>
<td>0.0189, 0.0544</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>258, 259</td>
<td>4.095, 3.691</td>
<td>0.1485, 0.1496</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>356, 357</td>
<td>3.211, 2.619</td>
<td>0.2511, 0.2521</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>437, 438</td>
<td>1.961, 1.242</td>
<td>0.3244, 0.3252</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>512</td>
<td>0.402</td>
<td>0.3925</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>91, 91</td>
<td>4.910, 4.688</td>
<td>0.0045, 0.0045</td>
<td>6085.45</td>
<td>6334.59</td>
</tr>
<tr>
<td></td>
<td>91, 146</td>
<td>4.535, 4.364</td>
<td>0.0045, 0.0344</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>179, 220</td>
<td>4.090, 3.789</td>
<td>0.0640, 0.1069</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>276, 276</td>
<td>3.404, 2.952</td>
<td>0.1682, 0.1682</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>370, 371</td>
<td>2.417, 1.810</td>
<td>0.2646, 0.2655</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>444, 445</td>
<td>1.127, 0.345</td>
<td>0.3302, 0.3310</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>