Journal of Computer Science 7 (9): 1439-1447, 2011
ISSN 1549-3636

Combined Heuristic Technique for Optimization of Bloom Filter in Spam Filtering

[1]Arulanand Natarajan, [2]S. Subramanian and [3]K. Premalatha
[1]Anna University of Technology, Coimbatore,
[2]Sri Krishna College of Engineering and Technology, Coimbatore
[3]Bannari Amman Institute of Technology, Erode, TN, India

Abstract: Problem statement: Spam is an irrelevant or inappropriate message sent on the internet to a large number of newsgroups or users. A spam word is a list of well-known words that often appear in spam mails. Bloom Filter (BF) is used for identification of spam word. **Approach:** BF is a simple but powerful data structure that can check membership to a static set. The trade-off to use BF is a certain configurable risk of false positives. The odds of a false positive can be made very low if the hash bitmap is sufficiently large. Bin Bloom Filter (BBF) has number of BFs which assign group of words into bins with different false positive rates based on weight of the spam words. Genetic Algorithm (GA) was employed to minimize the total membership invalidation cost of BBF. GA had premature convergence problem. Simulated Annealing (SA) was incorporated with GA to prevent the premature convergence effectively. **Results:** The experimental results of total membership invalidation cost are analyzed for various sizes of bins. The results showed that the combined GA-SA model outperforms SA and GA model. **Conclusion:** GA has premature convergence due to its genetic operators that are not able to generate offsprings which are superior to the parents. So more number of similar chromosomes presented on the population. When GA is incorporated with SA new genes were introduced which causes diversity in the population and prevents premature convergence. The combined GA-SA outperforms GA and SA.

Key words: Spam word, hash function, genetic algorithm, simulated annealing, static set, premature convergence, long vector, bit array

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 study 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 GA which minimizes the total membership invalidation cost. This study 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.

Corresponding Author: Arulanand Natarajan, Anna University of Technology, Coimbatore, TN, India

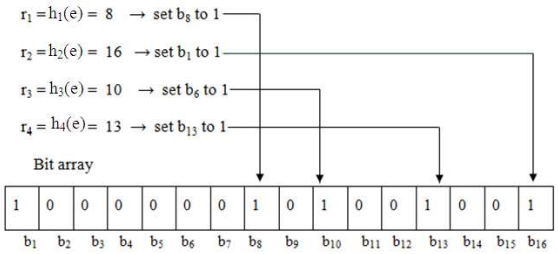

Fig. 1: Bloom filter

The organization of this study is as follows. Section 2 deals with the standard BF. Section3 presents the GA technique. SA is discussed in section 4. Section 5 explains the optimized BBF using GA with SA. Performance evaluation of the BBF of SA, GA and GA with SA are discussed in section 6.

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 Bloom filter 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, … , 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). Figure1 is an example where m=16, k=4 and e is the element to be stored in the bit array.

Bloom filters also have the unusual property that the time needed to either add items or to check whether an item is in the set is a fixed constant, O(k), completely independent of the number of items already in the set.

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

$$f = \left(1 - e^{-kn/m}\right)^k \quad (1)$$

Let:

$$g = k\ln(1 - e^{-kn/m})$$

Minimizing the false positive probability f is equivalent to minimizing with respect to k:

$$\frac{dg}{dk} = \ln\left(1 - e^{-\frac{kn}{m}}\right) + \frac{kn}{m}\frac{e^{-kn/m}}{1 - e^{-kn/m}}$$

The derivative equals 0 when k_{min}=(1n2)(m/n). In this case the false positive probability f is:

$$f(k_{min}) = (1 - p)^{k_{min}} = \left(\frac{1}{2}\right)^{k_{min}} = (0.6185)^{m/n}$$

Of course k should be an integer,

$$k = \lceil \ln 2.(m/n) \rceil \quad (2)$$

When the set of elements is changing over time, in that case the insertions and deletions in the Bloom filter become important. Inserting elements into a Bloom filter hash the element k times and set the bits to 1. However, deletion process is hashing the element to be deleted k times and set the corresponding bits to 0, is not possible. This is because setting a location to 0 that is hashed to by some other element in the set and the resultant Bloom filter is no longer correctly reflects all elements in the set. To avoid this problem, Counting Bloom Filter (CBF) has an entry in the Bloom filter is not a single bit but instead a small counter. When an item is inserted, the corresponding counters are incremented; and when an item is deleted, the corresponding counters are decremented.

Compressed Bloom filter (Mitzenmacher, 2002) improves the performance when the Bloom filter is passed as a message and its transmission size is a limiting factor. Bloom filter is suggested as a means for sharing Web cache information. In this setting, proxies do not share the exact contents of their caches, but instead periodically broadcast Bloom filters

representing their cache. By using compressed bloom filters, proxies can reduce the number of bits broadcast, the false positive rate and/or the amount of computation per lookup. The cost is the processing time for compression and decompression. It can use simple arithmetic coding and more memory use at the proxies, which utilize the larger uncompressed form of the Bloom filter.

The Spectral Bloom filter (Cohen and Matias, 2003*)* is an extension of the Bloom Filter to multi-sets, allowing the filtering of elements whose multiplicities are below a threshold given at query time. The spectral Bloom filter replaces the bit vector with a vector of m counters C. the counters in C roughly represent multiplicities of items. All the counters in C are initially set to 0. When inserting an item, it increases the counters $C_{h1(s)}$, $C_{h2(s)}$,…, $C_{hk(s)}$ by 1 and it stores the frequency of each item. It allows deletion by decreasing the same counters.

The one type of bloom filter is Split Bloom filter (Xiao *et al*., 2004). It increases the capacity by allocating a fixed s×m bit matrix instead of an m-bit vector as used by the Bloom Filter to represent a set. A certain number of s filters each with m bits, are employed and uniformly selected when inserting an item of the set. The false match probability increases as the set cardinality grows. The basic idea is, in element addition operation, before going to map element *x* into the standard bloom filter s, it first checks the bloom filters from 1 to s-1 whether they have response that element *x* is a member of set *A*. If the response is false, it makes sure that there is no false positive probability in first s-1 bloom filters, so it maps the element *x* into bloom filter s; otherwise, it just go ahead to the next element with no any operation on element *x*.

The Dynamic Bloom Filter (DBF) can support concise representation and approximate membership queries of dynamic set instead of static set (Guo *et al*., 2006). The basic idea of DBF is to represent a dynamic set with a dynamic s×m bit matrix that consists of s bloom filters. Here s is initialized to 1, but it is not a constant as split bloom filter. It can increase during the continuous increasing process of the set size.

The Scalable Bloom filter (Xie *et al*., 2007) represents dynamic data sets well and provides a way to effectively solve the scalability problem of Bloom filters. It solves the scalability problem of Bloom filters by adding Bloom filter vectors with double length when necessary.

The data structure of Hierarchical Counting Bloom Filter (Yuan *et al*., 2008) is composed of several sub CBFs. The number of these sub filters is h. Each sub filter has different counter length and bit array length. Each counter length is c_0, c_1… c_{h-1} and each bit array length is m_0, m_1, …, m_{h-1} respectively. $m_0 > m_1 > \ldots > m_{h-1}$.

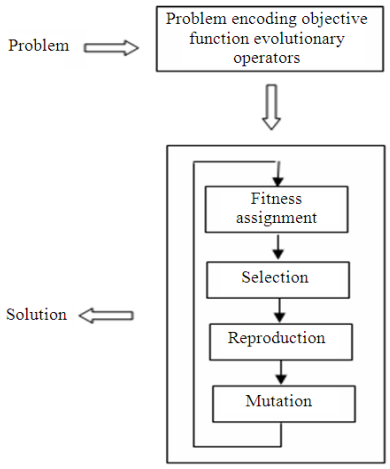

Fig. 2: Simple GA

Genetic algorithm: A Simple GA is a computational abstraction of biological GA proposed by Holland (1975), is a probabilistic optimal algorithm that is based on the evolutionary theories. This algorithm is population-oriented. Successive populations of feasible solutions are generated in a stochastic manner following laws similar to that of natural selection.

GAs is a family of computational models inspired by evolution. These algorithms encode a potential solution to a specific problem on a simple chromosome-like data structure and apply crossover and mutation operators to these structures so as to preserve critical information. An implementation of a GA begins with a population of (usually random) chromosomes. One then evaluates these structures and allocates reproductive opportunities in such a way that those chromosomes which represent a better solution to the target problem are given more chances to reproduce than those chromosomes which are poorer solutions. The goodness of a solution is typically defined with respect to the current population. Nazif and Lee (2010) proposed a GA using an optimized crossover operator designed by a complete undirected bipartite graph that finds an optimal set of delivery routes satisfying the requirements and giving minimal total cost for vehicle routing problem. Usually there are only two main components of GAs that are problem dependent: the problem encoding and the fitness function (objective function/evaluation function). A problem can be viewed as a black box with different parameters: The only output of the black box is a value returned by an evaluation function indicating how well a particular combination of parameter settings solves the optimization problem. The goal is to set the various parameters so as to optimize some output. In more traditional terms that to maximize (or minimize) some function $F(x_1,x_2,..., x_m)$. Figure 2 shows the Simple GA.

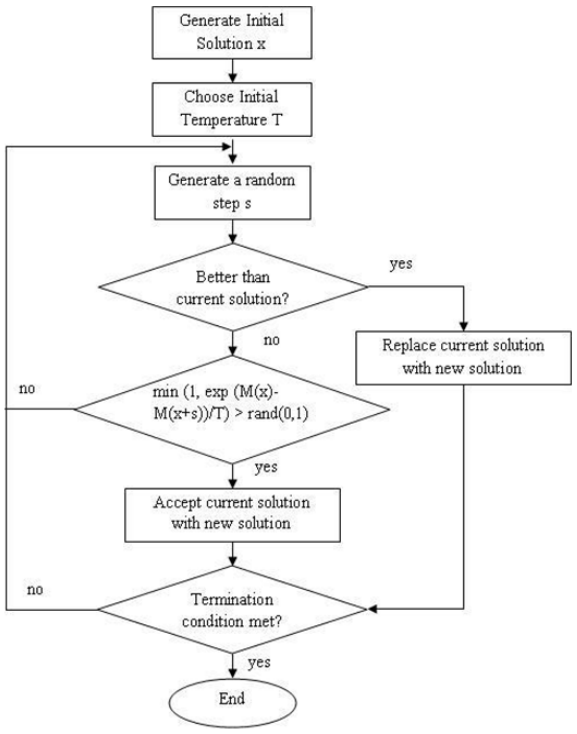

Fig. 3: Simulated annealing

Premature convergence problem: GA suffer from the premature suboptimal convergence or stagnation which occurs when some poor individuals attract the population due to a local optimum or bad initialization, it prevents further exploration of the search space (Bonabeau *et al*., 1999). One of the causes of this problem is that a very fit chromosome is generally sure to be selected for mating and since offspring resemble their parents, chromosomes become too similar. Hence, the population will often converge before reaching the global optimal solution, resulting in premature convergence. Also in GA the population size is finite, which influences the sampling ability of a GA and as a result affects its performance.

Incorporating a local search method can introduce new genes which can help to fight the genetic drift problem (Asoh and Muhlenbein 1994; Thierens *et al*., 1998) caused by the accumulation of stochastic errors due to finite populations. It can also accelerate the search towards the global optimum (Hart, 1994) which in turn can guarantee that the convergence rate is large enough to obstruct any genetic drift. In addition a local search method within a GA can improve the exploiting ability of the search algorithm without limiting its exploring ability (Hart, 1994). If the right balance between global exploration and local exploitation capabilities can be achieved, the algorithm can easily produce solutions with high accuracy (Lobo and Goldberg, 1997). The proposed work incorporates SA to GA to avoid premature convergence.

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 a d 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 algorithm 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 (-ΔE/T)), where ΔE 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. SA is inherently sequential and hence very slow for problems with large search spaces. Though a robust technique, its drawbacks include the need for a great deal of computer time for many runs and carefully chosen tunable parameters. The SA is used for water transfers (Khodabakhshi *et al.*, 2010) that consist of reservoirs and transfer systems. The results of this research indicated that the SA is capable of solving such complex problems in water resources management with good precision in a reasonable period of time.

Figure 3 shows the SA algorithm:

MATERIALS AND METHODS

Bloom filter optimization using GA-SA: Bin Bloom Filter (BBF): A BBF is a date 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 GA which minimize the total membership invalidation cost. Figure 4 shows Bin Bloom filter with its tuple <n,f,w> configuration.

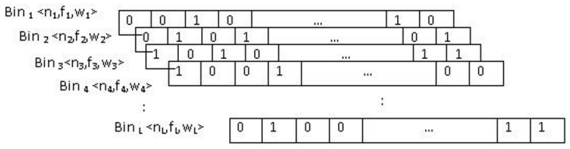

Fig. 4: Bin bloom filter

Problem definition: Consider a standard supervised learning problem with a set of training data D = {<Y_1,Z_1 >,..., <Y_i, Z_i>, … ,< Y_r ,Z_r >} , 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, Y2, … , Y_r 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_L}, {w_1, w_2,…, w_L}, 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:

$$F= n_1f_1w_1 + n_2f_2w_2 +……+ n_Lf_Lw_L$$

The total membership invalidation cost

$$F(L) = \sum_{i=1}^{L} n_1 w_i f_i \quad (3)$$

To be minimized:

Where:

$$\sum_{i=1}^{L} n_i = N$$

N- Total number of Strings in a spam set:

$$f_i = \left(\frac{1}{2}\right)^{\ln 2 \times \left(r_i m \Big/ \sum_{j=1}^{i} n_j r_j\right)}$$

$$r_i = \ln(f_i)(1 \le i \le L)$$

The objective function f(L) taken as standard for the problem of minimization is:

$$f(L) = \begin{cases} C_{max} - F(L) & \text{if } F(L) < C_{max} \\ 0 & \text{if } F(L) \ge C_{max} \end{cases} \quad (4)$$

where, C_{max} is a large constant.

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:

$$P_s = \frac{\frac{f_s}{N_s}}{\frac{f_{ns}}{N_{ns}} + \frac{f_s}{N_{ns}}} \quad (5)$$

$$P_{ns} = \frac{\frac{f_{ns}}{N_{ns}}}{\frac{f_{ns}}{N_{ns}} + \frac{f_s}{N_{ns}}} \quad (6)$$

P_s = The probability of a word given the mail is spam
P_{ns} = The probability of a word given the mail is legitimate
f_s = The frequency of word in the spam documents
f_{ns} = Frequency of words in the legitimate documents
N_s = The total spam documents
N_{ns} = 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:

$$\text{weight}_{word} = \frac{P_s}{P_{ns}} \quad (7)$$

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.

Chromosome representation: 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 is shown in Figure 5.

where, $n_{ij,}$ f_{ij} and w_{ij} refer respectively the number of words, false positive rate of and the weight of the jth bin of i^{th} chromosome. A set of 3 genes <n,f,w> encodes a protein-a trait, that is a single bin. The false positive rate f_{ij} can be obtained from Eq. 1 where n_{ij} is drawn from the gene of the chromosome, m is known in advance and k is calculated from Eq. 2.

Initial population: One chromosome in the population represents one possible solution for assigning the triples <n, f, w> for L Bloom filters. Therefore, a population represents a number of candidate solutions for the Bloom filters. At the initial stage, each chromosome randomly chooses different <n, f, w> for L Bins. The fitness function for each individual can be calculated based on the Eq. 4.

Selection: In selection the offspring producing individuals are chosen. Each individual in the selection pool receives a reproduction probability depending on the own fitness value and the fitness value of all other individuals in the selection pool. This fitness is used for the actual selection in the step afterwards. This simplest selection scheme is roulette-wheel selection, also called stochastic sampling with replacement. The proposed system employs roulette-wheel selection method.

Crossover: The interesting behaviour happens from GAs because of the ability of the solutions to learn from each other. Solutions can combine to form offspring for the next generation. Occasionally they will pass on their worst information, but doing crossover in combination with a powerful selection technique perceives better solutions result. Crossover occurs with a user specified probability called, the crossover probability Pc. Many crossover techniques exist for individual. In single point crossover, a position is randomly selected at which the parents are divided into two parts. The parts of the two parents are then swapped to generate two new offspring.

Mutation: Mutation is a genetic operator that alters one or more gene values in a chromosome from its initial state. This can result in entirely new gene values being added to the gene pool. With these new gene values, the GA may be able to arrive at better solution than was previously possible. Mutation applied usually with a low probability to introduce random changes into the population. It replaces gene values lost from the population. It helps to prevent the population from stagnating at any local optima and it avoids premature convergence. Mutation evaluates more regions of the search space that is it makes the entire search space reachable. It occurs during evolution according to a user-definable mutation probability. This probability should usually be set fairly low. If it is set to high, the search will turn into a primitive random search.

Evaluation: After producing offspring they must be inserted into the population. By a reinsertion scheme individuals should be inserted into the new population and it determines which individuals of the population will be replaced by offspring. The used selection algorithm determines the reinsertion scheme. The elitist combined with fitness-based reinsertion prevents this losing of information and is the recommended method. At each generation, a given number of the least fit parent is replaced by the same number of the fit offspring.

Combined GA and SA: A combination of a GA and a SA can speed up the search to locate the exact global optimum. In this hybrid, applying a SA to the solutions that are guided by a GA to the most promising region can accelerate convergence to the global optimum. The time needed to reach the global optimum can be further reduced if local search methods and local knowledge are used to accelerate locating the most promising search.

For any hybrid algorithm, a local search can be applied to either every individual in the population or only few individuals. Applying a local search to every individual in the population on expensive function evaluations can waste resources without providing any more useful information. Applying a local search to a large fraction of the population can limit exploration of the search space by allowing the GA to evolve for a small number of generations. Deciding upon the optimal fraction of the population which should perform local search and the basis on which these individuals are chosen, has a great impact on the performance of a hybrid. The proposed system incorporates SA in GA for every 10 iteration of 10% population.

RESULTS

In the proposed system the roulette wheel selection, objective function value as fitness, single point crossover, uniform mutation and maximum iteration numbers as stopping criterion are used in the experimental analysis.. The single point crossover is applied with a probability of P_c. For every bit of the string, mutation occurs with probability P_m. The levels of operator probabilities are drawn from the literature. Suggested that crossover rates between 0.65 and 1 and mutation rates between 0.001 and 0.01 are useful in GA applications. Population size and maximum generation number have also positive effects on finding the best fitness value.

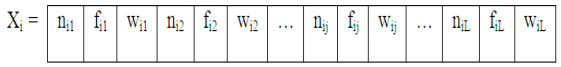

Fig. 5: Chromosome representation for bin bloom filter

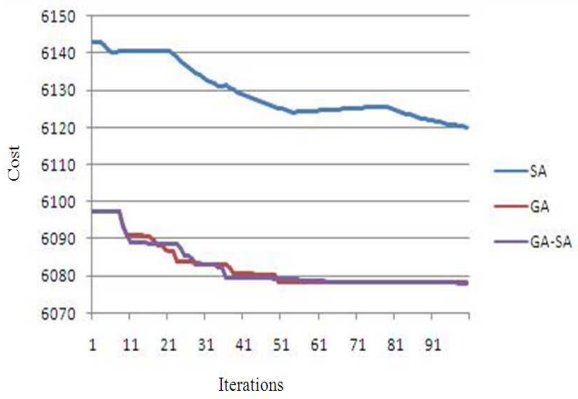

Fig. 6: Total membership invalidation cost for bin size 10

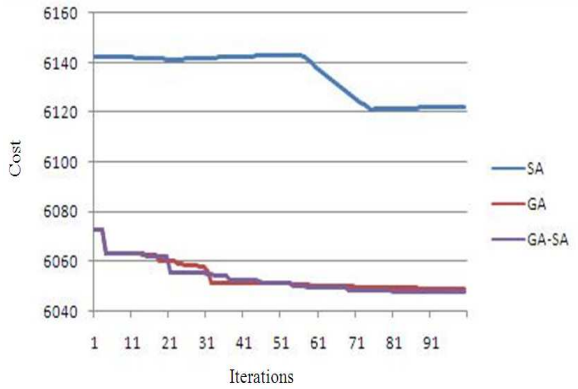

Fig. 7: Total membership invalidation cost for bin size 11

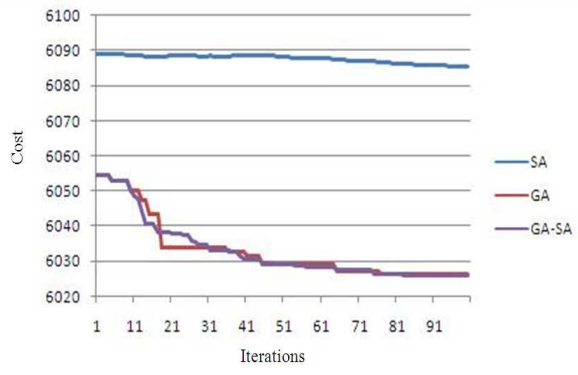

Fig. 8: Total membership invalidation cost for bin size 12

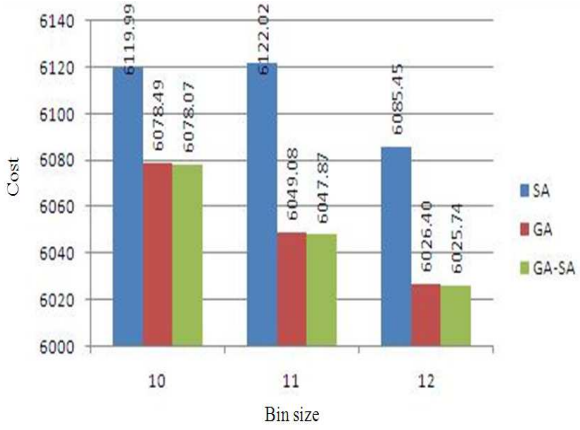

Fig. 9: Performance evaluation chart for SA, GA and hybrid GA-SA

Increasing the population size or number of generations enlarges the search space. At the end of the analysis, the crossover probability level P_c as 0.65, mutation level P_m as 0.01, population size as 100 and maximum number of iteration is set as 100. The total number of strings taken for testing is 3000 and their weights are ranging from 0.0005-5. The size of the BF is 1024. The highest temperature T is assigned as 1024, the BF size. These experimental values are tested for bin sizes from 10-12.

Since BF allows false positive, the membership invalidation cost is unavoidable. For BBF, the total membership invalidation cost is expressed in (3). In standard BF, different weights in different bins into consideration, the total membership invalidation cost is then as follows:

$$F_{standard} = (n_1w_1 + n_2w_2 + \ldots\ldots + n_Lw_L)f$$

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

Figure 6-8 show the total membership invalidation cost attained from SA, GA, combined GA and SA (GA-SA). They illustrate that there is a noticeable variation between SA and variants of GA such as GA and GA-SA. And also they show that GA-SA outperforms SA and GA.

Table 1-3 show the values obtained from SA, GA, GA-SA respectively. The first column represents the bin size; the second column, number of strings in each bin; the third column, average weight of the strings present in a bin; the fourth column, false positive rate of each bin. The fifth column shows the total membership invalidation cost of BBF. Figure 9 shows the evaluation chart for the cost obtained from SA, GA and GA-SA.

Table 1: Values obtained from SA

Bin size	Number of strings	Average String weight	False positive rate	Cost of BBF
10	96, 129. 174, 212	4.906, 4.663, 4.424, 4.101	0.0059, 0.0221, 0.0592, 0.0982	6119.99
	319, 321, 383, 385, 490, 491	3.694, 3.190, 2.607, 1.966	0.2139, 0.2159, 0.2767, 0.2786	
		1.250, 0.385	0.3664, 0.3671	
11	45, 45,124, 169	4.956, 4.757, 4.677, 4.445	1.79E-05, 1.79E-05, 0.0189, 0.0544	6122.02
	258, 259, 356, 357	4.095, 3.691, 3.211, 2.619	0.1485, 0.1496, 0.2511, 0.2521	
	437, 438, 512	1.961, 1.242, 0.402	0.3244, 0.3252, 0.3825	
12	91, 91, 91, 146	4.910, 4.688, 4.535, 4.364	0.0045, 0.0045, 0.0045, 0.0344	6085.45
	179, 220, 276, 276	4.090, 3.789, 3.404, 2.952	0.0640, 0.1069, 0.1682, 0.1682	
	370, 371, 444, 445	2.417, 1.810, 1.127, 0.345	0.2646, 0.2655, 0.3302, 0.3310	

Table 2: Values obtained from GA

Bin size	Number of strings	Average string weight	False positive rate	Cost of BBF
10	101, 113, 133, 156	4.901, 4.662, 4.465, 4.230	0.0077, 0.0129, 0.0247, 0.0427	6078.495
	210, 290, 466, 508	3.938, 3.558, 2.954, 2.135	0.0961, 0.1833, 0.3479, 0.3797	
	511, 512	1.306, 0.402	0.3818, 0.3825	
11	91, 110, 113, 130	4.910, 4.681, 4.496, 4.306	0.0045, 0.0114, 0.0129, 0.0227	6049.081
	149, 180, 246,445	4.068, 3.821, 3.498, 2.944	0.0368, 0.0650, 0.1353, 0.3310	
	512, 512,512	2.140, 1.307, 0.402	0.3825, 0.3825, 0.3825	
12	91, 93, 96, 108	4.910, 4.688, 4.530, 4.374	0.0045, 0.0050, 0.0059, 0.0105	6026.399
	130, 139, 168, 211	4.174, 3.962, 3.732, 3.440	0.0227, 0.0290, 0.0535, 0.0971	
	438, 502, 512, 512	2.922, 2.132, 1.307, 0.402	0.3252, 0.3753, 0.3825, 0.3825	

Table 3: Values obtained from GA-SA

Bin size	Number of strings	Average string weight	False positive rate	Cost of BBF
10	105, 120, 132, 139	4.898, 4.652, 4.448, 4.224	0.0092, 0.0166, 0.0241, 0.0290	6078.068
	207, 299, 462, 512	3.951, 3.567, 2.959, 2.140	0.0929, 0.1929, 0.3448, 0.3825	
	512, 512	1.307, 0.402	0.3825, 0.3825	
11	93, 103, 114, 123	4.908, 4.681, 4.503, 4.317	0.0050, 0.0084, 0.0134, 0.0183	6047.868
	152, 161, 237, 482	4.084, 3.845, 3.544, 2.975	0.0393, 0.0471, 0.1254, 0.3603	
	511, 512, 512	2.139, 1.307, 0.402	0.3818, 0.3825, 0.3825	
12	87, 88, 90, 111	4.913, 4.696, 4.547, 4.396	0.0035, 0.0037, 0.0042, 0.0119	6025.74
	114, 149, 158, 202	4.206, 3.997, 3.766, 3.489	0.0134, 0.0368, 0.0444, 0.0875	
	480, 499, 511,511	2.950, 2.126, 1.304, 0.401	0.3588, 0.3731, 0.3818, 0.3818	

DISCUSSION

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 false positive rate of less significant words. GA has been used many types of optimization problem. Premature convergence was the main problem for GA. It is caused by lower diversity of the population. Apparently maintaining higher diversity is important to obtained better result. To increase the diversity as well as preventing premature convergence SA is incorporated with GA for every 10 iterations of 10% of population. The experiment results show that the results obtained from GA-SA have lesser total membership invalidation cost than values obtained from SA and GA.

REFERENCES

Asoh, H. and H. Mühlenbein, 1994. On the mean convergence time of evolutionary algorithms without selection and mutation. Parallel Problem Solving Nat., PPSN III, 866: 88-97. DOI: 10.1007/3-540-58484-6_253

Bloom, B., 1970. Space/time tradeoffs in hash coding with allowable errors. Commun. ACM., 13: 422-426. Doi: 10.1145/362686.362692

Bonabeau, E., M. Dorigo and G. Theraulaz, 1999. Swarm Intelligence from Natural to Artificial Systems. 1st Edn., Oxford University Press, New York, ISBN0195131584, pp: 307.

Cohen, S. And Y. Matias, 2003. Spectral bloom filters. Proceeding of the ACM SIGMOD International Conference on Management of Data, (SIGMOD '03), ACM New York, USA, pp: 241-252. DOI: 10.1145/872757.872787

Feng, W., K.G. Shin, D.D. Kandlur and D. Saha, 2002. The BLUE active queue management algorithms. IEEE/ACM Trans. Network., 10: 513-528. DOI: 10.1109/TNET.2002.801399

Goldberg, D.E., 1989. Genetic Algorithms-in Search, Optimization and Machine Learning, Addison-Wesley Publishing Company Inc., Reading, Mass., ISBN0201157675, 9780201157673, pp: 412.

Guo, D., J. Wu, H. Chen and X. Luo, 2006. Theory and network applications of dynamic bloom filters, Proceeding of the 25th IEEE International Conference on Computer Communications, April 23-29, IEEE Xplore Press, Barcelona, Spain, pp: 1-12. DOI: 10.1109/INFOCOM.2006.325

Hart, W.E., 1994. Adaptive global optimization with local search. Ph.D. thises, University of California San Diago.

Holland, J., 1975. Adaption in Natural and Artificial Systems. 4th Edn., U Michigan Press, Oxford, England, pp: 183.

Khodabakhshi F, A. R. Ghirian and N. Khakzad , 2009. Applying Simulated Annealing For Optimal Operation of Multi-Reservoir Systems . Am. J. Eng. Applied Sci., 2: 80-87. DOI: 10.3844/ajeassp.2009.80.87.

Lobo, F.G. and D.E. Goldberg, 1997. Decision making in a hybrid genetic algorithm. Proceeding of the IEEE International Conference on evolutionary Computation, APR. 13-16, IEEE Xplore Press, USA, pp: 121-125. DOI: 10.1109/ICEC.1997.592281

Mitzenmacher, M., 2002. Compressed bloom filters. IEEE/ACM Trans. Network., 5: 604-612. DOI: 10.1109/TNET.2002.803864

Nazif H and L. S. Lee , 2010. Optimized Crossover Genetic Algorithm for Vehicle Routing Problem with Time Windows. Am. J. Applied Sci., 7: 95-101. DOI: 10.3844/ajassp.2010.95.101.

Thierens, D., D. Goldberg and P. Guimaraes, 1998. Domino convergence, drift and the temporal-salience structure of problems. Proceeding of the IEEE International Conference on Evolutionary Computation Anchorage, May 4-9, IEEE Xplore Press, USA, pp: 535-540. DOI: 10.1109/ICEC.1998.700085

Xiao, M., Y. Dai and X. Li, 2004. Split bloom filter. Chinese J. Electronic, 32: 241-245.

Xie, K., Y. Min, D. Zhang, J. Wen and G. Xie, 2005. Basket bloom filters for membership queries, Proceedings of IEEE Reigon 10, Nov. 21-24, IEEE Xplore Press, USA, pp: 1-6. DOI: 10.1109/TENCON.2005.301258

Xie, K., Y. Min, D. Zhang, J. Wen and G. Xie, 2007. A Scalable bloom filter for membership queries. Proceeding of the IEEE Conference on Global Telecommunications, IEEE Xplore Press, USA, pp: 543-547. Doi: 10.1109/GLOCOM.2007.107

Yuan, Z., Y. Chen, Y. Jia and S. Yang, 2008. Counting evolving data stream based on hierarchical counting bloom filter. Proceeding of the International Conference on Computational Intelligence and Security, Dec. 13-17, IEEE Xplore Press, USA, pp: 290-294. DOI: 10.1109/CIS.2008.216