A Bio Inspired Alternative to Huffman Coding

Natalia TATAR, Stefan HOLBAN
"Politehnica" University of Timisoara
Pta. Victoriei nr. 2, RO-300006 Timisoara
tatar.natalia@gmail.com , stefan@cs.utt.ro

***Abstract* — In the domain of data compression, the Huffman algorithm is one of the most used and referenced algorithm. Since it was published in 1952 by its author David A. Huffman until today, it is still used in a large area of fields. It was constantly changed during the time to improve performance in accordance to the field demands. We are proposing in this article a new approach to generate the Huffman codes and also to encode and decode data information. The proposed Huffman algorithm will use the Artificial Bee Colony algorithm to generate the codes. It shows that by using the bio inspired algorithm, the performance of the encode process is significantly better. By this approach we eliminate the drawbacks of using the tree data structure to generate the Huffman codes in terms of lookup time by using the Artificial Bee Colony.**

***Index Terms* — algorithms, artificial intelligence, data compression, data structures, Huffman codes.**

I. INTRODUCTION

In the field of data compression, the Huffman algorithm and the Huffman codes are frequently used. Since it was published until today, the algorithm was constantly changed. Its applications can vary from simple compression tools to integration into network transfer frameworks. All of the encoding algorithms use a binary tree to generate the Huffman codes. Using this type of structure draws its effects and weaknesses, such as high lookup time within the tree structure and time to generate the codes.

In this article we will highlight and take into discussion one of the most pressing issues related to the generation methods of the Huffman codes. As already stated, all of the previous methods use the data structure of binary trees. If the data to be encoded is large, the binary tree will also be ample. The way the binary tree is constructed is also not optimal. The actual data is contained only in the leaf nodes and all the parent nodes are there to help generate the code, but do not hold any actual relevant data.

There are researches in this direction, to replace the binary tree structure. In [1] they attempt to replace the structure with arrays. The search continues to bring improvements to this algorithm as in [2] authors show that the compression ratio can be improved by up to 12 % if the data is encoded in subgroups, such as a set of codes for alphabet and another set of codes for number or other special characters. Additional, the Huffman algorithm was researched in embedding secret messages in text files or videos [3]. The authors of [5] show that using Huffman coding can prolong the life of wireless sensor networks (WSN). Almost 80% of such networks is used on data transmission, developing a better and efficient compression algorithm can bring significant improvement.

Also other fields study the usage of this algorithm, such as [4] network intrusion tracking and prevention or in the area of test data of IP cores (Intellectual Property) [6].

Improving the Huffman algorithm and the method to generate the Huffman codes, can bring great results in a large area of domains.

In this article we will present a new method to generate the Huffman codes and the new encode and decode algorithms. We will make use of the Artificial Bee Colony, a population based bio-inspired algorithm, to randomly generate a set of codes which we will enforce to poses Huffman coding properties. By this approach we achieve a significant improvement in the Huffman algorithm by eliminating the lookup time for a symbol as a means to create its Huffman code.

II. HUFFMAN ALGORTHM

Huffman algorithm is an entropy encoding algorithm used for lossless data compression [7]. Lossless data compression specific feature is that by the encoding/decoding procedure there is no data loss between the original file and the resulting file.

It uses variable-length codes to encode a symbol in the source file, so that the most frequent symbol will get the shortest code. The codes are created by using a binary tree structure by using the symbols frequency. The tree structure is constructed by starting with the less frequent symbols and builds until all symbols are added to the structure as leaf nodes.

Symbol	Times	Frequency	Entropy	Codes	A	B
G	8	0.2	2.321928095	00	64	16
F	7	0.175	2.514573173	111	56	21
E	6	0.15	2.736965594	110	48	18
D	5	0.125	3	100	40	15
space	5	0.125	3	101	40	15
C	4	0.1	3.321928095	010	32	12
B	3	0.075	3.736965594	0111	24	12
A	2	0.05	4.321928095	0110	16	8
Total	40		24.95428865		320	117

Table 1. A set of symbols and values

For example, if we have the symbols and frequencies from Table 1, we will get the binary tree from Figure 1. In Table 1, the columns A and B represent the number of bits that will require to store the text if not encoded (column A), using ASCII characters, and number of bits required if the text is encoded with Huffman codes(column B). With these values, the compression ration of our example would be 63.4375 according to the formula in Equation [1].

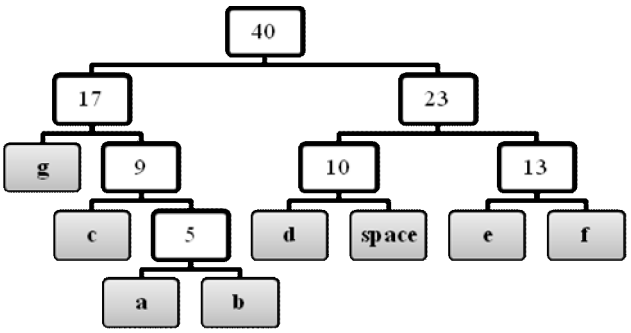

Figure 1. Huffman tree structure for Table 1

$$C = \left(1 - \frac{compressed\_size}{raw\_size}\right) * 100;$$

Equation [1]

We can see the only leaf nodes contain the symbols. The coding of the symbols begins from the root to the leaf, where a left gives 0 and a right gives 1. The entropy in Table 1 is calculated by the following formula:

$$e_i = -\log_2 \frac{f_i}{n};$$

Equation [2]

Where:

e_i – Entropy of ith symbol

f_i – Frequency of ith symbol

n – Total number of symbols

The entropy calculates how much information is encoded in a message. The entropy of a message is the sum of all its symbols. The higher the entropy of a message, the more information it contains. [14]

There are different variations to the Huffman algorithm like the Adaptive Huffman code. The enhancement consists in the fact that the source file is only once traversed. The Huffman tree is updated constantly after each symbol is encoded, but keeping the sibling property of the tree and also the correct weight of the nodes by swapping the correct ones.

III. ARTIFICIAL BEE COLONY

The Artificial Bee Colony algorithm (ABC) is a population based algorithm that was introduced by Karaboga in 2005. It is inspired by the honey bee behavior of foraging [8].

The algorithm consists or three components: employed bees, unemployed bees and food sources. The employed bees search for the most profitable food source, depending of closeness to the hive and especially of how much food it consists. The algorithm is mostly used as an optimization algorithm, for example its first application was solving problems such as the job scheduling problem [9] or traveling salesman [10].

In nature, the bee foraging activity begins with the scouts. They will scout the area around the hive to find food sources. Scouts can search to a radius of up to three kilometers from the hive. After finding the sources of food, they return to the hive and share the information. They do that by performing a dance (waggle dance). Depending on the length and other properties of the dance, they manage to share information on the food sources, like distance from the hive in coordination with the sun position and how much nectar it holds. From the dance area, other working bees can select a food source to collect nectar from. The hive always has some scouts to keep looking for new food sources.

As in nature, the algorithm keeps the main elements. As proposed in [11] there are three kinds of bees: workers, onlookers and scouts. Worker bees are the ones who visit a food source that it was previously visited by itself, the onlookers are the ones who check the dance area to decide the food patch that they will visit and the scouts are the bees that keep searching for new sources around the hive. The algorithm proposed uses the half of the hive bees as worker bees. Also every working bee has its own food patch, so the number of worker bees equals the number of food sources. After a working bee has depleted its source, it becomes an onlooker. The main steps of the algorithm proposed are as follow:

•Initialize.
• REPEAT.
  (a) Place the employed bees on the food sources in the memory;
  (b) Place the onlooker bees on the food sources in the memory;
  (c) Send the scouts to the search area for discovering new food sources.
• UNTIL (requirements are met).

At the initial stage a random number of food sources are selected and the bees determine their nectar amount. This bees return to the hive and perform the dance for the onlookers. Secondly, after they shared the information, every employed bee visits the food patches they previously visited, as that is in its memory, and choose new food sources in the neighborhood of the one they visit. If an onlooker chooses a food area depending on the information provided by the workers through the dance, it will visit the sources in its neighborhood. If a source is abandoned it will be replaced by a new one that was previously found by the scout bees. In this proposed model by [11], at each iteration the scout bees find at least one new food source.

By food source we describe an actual possible solution. The number of total food sources is actually the total number of possible solutions for the problem. At every iteration we memorize the best solution that was found until some criteria are met or the number of total iterations was reached.

There are also other approaches to the algorithm steps, but they keep the main elements. For example in paper [12] and [13] there is a distinction in the algorithm steps, dividing the steps into forward step and backward step of the bee activity.

The algorithm can give a set of possible solutions to a certain problem by memorizing the last few best solutions.

IV. ENCODING/DECODING PROCEDURES

The encoding procedure differs by whole in comparison with the classic approach. For beginnings we will not build a binary tree to generate the Huffman codes. By using the ABC algorithm we can generate them without it.

First we need to be aware that there are some fundamental properties of the Huffman codes. We will generate the appropriate code set to be assigned to the encoded data by always forcing the set to have the following properties:

1. The prefix property – no other code is prefix to any other code in the set.
2. The codes must be minimal.
3. The shortest code will be assigned to the most frequent symbol.

We will use the ABC algorithm to generate the code set. In the forward pass, the artificial bee will generate a new code that was not used before. Also along with the code it will choose the code length.

In the backward pass, the bee will return to the hive and the prefix property of the code set will be enforced, meaning that the bees that do not comply will have to search for a new code.

Once the code set was established, each code will be assigned to the data symbols (Figure 2). This process will take place according to their frequencies.

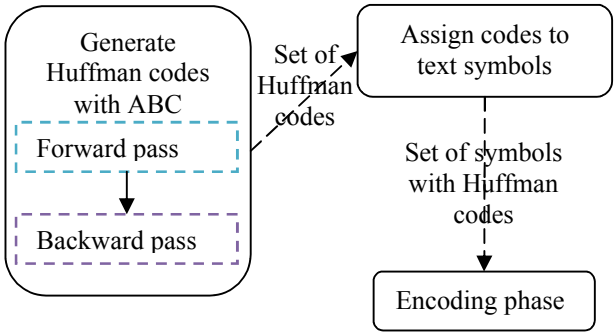

Figure 2. Chart of assigning Huffman codes.

The process is simple and straightforward. Due to the ABC proprieties, the algorithm implemented should have significant performance improvement. We do not have any unnecessary data stored and processed.

The following pseudo code shows the forward and backward pass to generate the Huffman code set.

```
procedure Forwardpass()
begin
for i=1 to bees
  if       code_not_assigned(i)       then
  genetrate_random_number(i,255);
    while (true)
       generate_random_mask();
       if mask_is_acceptable() then
          break;
       endif
    endwhile
   endif
assign_code;
endfor
call Backwardpass();
end.

procedure Backwardpass()
begin
for i=1 to bees
  for j=1 to bees
    if code(i)<code(j)
     if prefix(code(i),code(j))==TRUE then
        code(j)=0;
  endfor
endfor
  for i=1 to bees
       if code(i)==0 then
       call Forwardpass();
  endfor
  end.
```

In the forward pass procedure the main variable is bees, which designates the number of artificial bees in the hive of the ABC algorithm. In our article the number of bees equals the number of symbols in the text. Each symbol will have a randomly generated number and mask. The number range for the code is {0 … 255} (which is in accordance to the ASCII code) and for the mask is {2 … 8}. In the mask interval we have excluded the number 1 because we can not have a code with just one bit. We need to generate additional to the number also a mask because the Huffman algorithm uses variable length codes in terms of bit number and the smallest unit we can address is a byte.

In the backward pass procedure we will check the set of codes and enforce the prefix property. We will reset the codes with the higher mask and call the forward pass procedure, where this code will again generate random number and mask if the prefix property is not followed. This routine calls between forward pass and backward pass will stop when we have a set that meets the requirements we have set.

The decoding process is basically the same as the classic approach. Each code can be decoded one at the time, the prefix property ensures that the decode process will be done successful and without misinterpretation.

V. EXPERIMENTAL RESULTS

We have tested the encode process against the classic approach of the Huffman coding.

The program main procedures are as described in the article. After the file was read and the symbols and they frequency were established, the ABC procedure is called. This procedure searches for a set of codes with Huffman properties. It then assigns the codes to the symbols according to their frequency; the most frequent symbols get the shortest codes.

Series	Nr. Symbols	ABC Huffman	Classic Huffman
1	25	0.001	0.149
2	26	0.001	0.117
3	57	0.002	0.568
4	61	0.002	0.488
5	65	0.001	0.861

Table 2. Huffman comparison table

We have compared the time needed for our algorithm to generate and assign codes to the classic Huffman algorithm. Also, in this article we do not discuss the compression rate of the files, as we compare fundamentally the same technique.

In Table 2 we have a set of values representing the runtime of the respective algorithms. The computer used to generate this values has a processor installed Intel64 Family 6 Model 37 Stepping 2 Genuine Intel 2267 MHz. We have measured the time in clocks/sec.

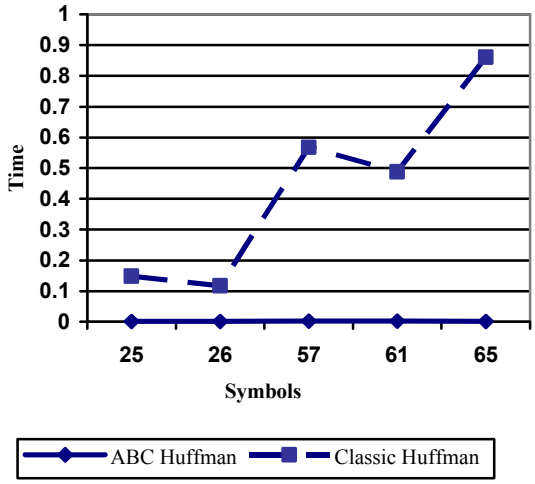

Figure 3 Results of implementation

We can easily observe that the time needed for our new proposed algorithm is fairly lower then the classic Huffman algorithm. This can be explained by the fact that we do not use additional structure to generate the codes, but we generate them randomly. The set of codes is checked that it corresponds with the properties of a Huffman code. It has to follow the prefix property first and then the codes are assigned to the symbols depending of their frequency, the most frequent symbol will get the shortest code.

VI. CONCLUSION

The results can be explained by the fact that generating the code set is not a very time consuming task, because it is not highly affected by the number of symbols. On the other hand, the classic Huffman implementation is proportional influenced by this number. The higher the number, the lower is the performance of the codes generated in terms of time.

We have presented a bio-inspired approach to generate the Huffman codes and also the encoding process using ABC algorithm. The experimental results show improvement in the time needed to generate the codes.

REFERENCES

[1] Wang Pi-Chung, "A Memory-efficient Huffman Decoding Algorithm",The 19th International Conferance on Advanced Information Networking and Applications, pp. 475-479, 2005.

[2] Shukala Piyush Kumar, "Multiple Subgroup Data Compression Technique Based on Huffman Coding", First International Conferance on Computational Intelligence, Communication Systems and Networks, pp. 397-402, 2009.

[3] Chen Kuo-Nan, "Embedding Secret Messages Using Modified Huffman Coding. ", Fifth International Conferance on Inteligent Information Hiding and Multimedia Signal Processing, pp. 278-281, 2009.

[4] Zheng Ruijuan, Wu Qingtao and Zhang Mingchuan. "An Intelligent Packet Marking Algorithm based on Extended Huffman Coding.", Second International Symposium on Inteligent Information Technology Application, pp. 60-64, 2008.

[5] Yuanbin Mo, "A data compretion algorithm based on adaptive Huffman coding for Wireless Sensor Networks." Forth International Conferance on Intelligent Computation Technology and Automation, pp. 3-6, 2011.

[6] Kavousianos X., Kalligeros, E. and Nikolos, D. ,"Efficient Test Data Compression for IP Cores using multilevel Huffman Coding.", Design, Automation and Test in Europe, 2006. DATE '06. Proceedings , pp. 1-6, 2006. 3-9810801-1-4 .

[7] Wikipedia. Huffman coding. [Online] [Cited: 02 08,2012.] Available: http://en.wikipedia.org/wiki/Huffman_algorithm

[8] Karaboga Dervis. "Artificial bee colony algorithm." [Online] 5(3):6915, 2010. [Cited: February 08, 2012.] Available: http://mf.erciyes.edu.tr/abc/publ.htm.

[9] Chong Chin Soon, "A Bee Colony Optimization Algorithm to job shop scheduling.", Winter Simulation Conference, pp. 1954-1961, 2006,.

[10] Wong Li-Pei, Low Malcolm Yoke Hean and Chong Chin Soon, "A Bee Colony Optimization Algorithm for Traveling Salesman Problem.", Second Asia International Conference on Modelling & Simulation, pp. 818-823, 2008.

[11] Karaboga Dervis and Basturk Bahriye, "A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. ", Journal of Global Optimization, pp. 459-471, 2007.

[12] D.T. Pham, A. Ghanbarzadeh, E. Koç, S. Otri , S. Rahim , M. Zaidi. "The Bees Algorithm – A Novel Tool for Complex Optimisation." , In Proceedings of the 2nd International Virtual Conference on Intelligent Production Machines and Systems (IPROMS 2006), pp. 454-459, 2006.

[13] Dušan Teodorovic, Mauro Dell' Orco. "Bee colony optmization– A cooperative learninig.", In Proceedings of the 16th Mini-EURO Conference on Advanced OR and AI Methods in Transportation , pp. 51-60 , 2005

[14] Mark Nelson, Jean-Loup Gailly – "The Data Compression Book", Second edition, M&T Books, New York, pp 16, 1995, 1558514341.