A METHOD FOR COMPRESSION OF SHORT UNICODE STRINGS

Masoud Abedi
Rayan Tahlil Sepahan Co.,
Isfahan Science and Technology Town, Isfahan
Iran
Abbas Malekpour
Faculty of Computer Science,
University of Rostock,
Germany

Peter Luksch
Faculty of Computer Science,
University of Rostock,
Germany
Mohammad Reza Mojtabaei
Department of Computer,
Mohajer Technical University, Isfahan,
Iran

Abstract—The use of short texts in communication has been greatly increasing in recent years. Applying different languages in short texts has led to compulsory use of Unicode strings. These strings need twice the space of common strings, hence, applying algorithms of compression for the purpose of accelerating transmission and reducing cost is worthwhile. Nevertheless, applying the compression methods like gzip, bzip2 or PAQ due to high overhead data size at the beginning of the message is not appropriate. The Huffman algorithm is one of the rare algorithms effective in reducing the size of short Unicode strings. In this paper, an algorithm is proposed for compression of very short Unicode strings. This proposed algorithm has four stages for compression. In the first stage, every new character to be sent to a destination is inserted in the proposed mapping table. At the beginning, every character is new. In case the character is repeated for the same destination, it is not considered as a new character. In the second and third stages, the new characters together with the mapping value of repeated characters are arranged through a specific technique. In the fourth stage, the characters of stage three are specially formatted to be transmitted. The results obtained from an assessment made on a set of short Persian and Arabic strings indicate that this proposed algorithm outperforms the Huffman algorithm in size reduction.

Keywords—Algorithms, Data compression, Decoding, Encoding, Huffman codes, Text communication

I. INTRODUCTION

As the internet is growing significantly all over the world, the speed of data transmission needs to be increased day by day. Strings are used widely in many areas such as short messages, social networks, webpages, and in distributed databases. The number of webpages is on a drastic rise globally. It was shown that the number of available webpages are doubled every 9 to 12 months [1]. On the other hand, the use of social networks is on a constant intense growth. Facebook as the most famous social network in 2014, recorded a number of 1.4 billion as the number of its active users in one month. Facebook has evaluated the number of its mobile users by 90% [2].Meanwhile, the role of smartphones should not be ignored. Considering the great number of strings in the internet, the
necessity to reduce the costs and increase the speed of data transmission is an endless challenge [1]. The need to support other languages increases the use of Unicode standard, which in turn would double the strings’ size. For example, for sending a short message through mobile phone, 16 bits of space is used for transmission of a single character applying 2-byte Universal Character Set (UCS2), while it takes up only 8 bits without the use of Unicode. Therefore, using Unicode (UCS2) would reduce the message length from 140 characters to 70 characters [3].

The string size appears to be very low but considering the great number of messages sent and received in a day only by a specific user of a social network, it would be a substantial size. Regarding Short Message Service (SMS), the number of sent messages only in December 2009 reached 152.7 billion through 286 million users; which means an average of 534 messages per user in one month [4]. Consider a distributed database with millions or maybe billions of records; how many strings should be sent and received for creating such a database and keeping it up-to-date? This extensive application justifies the importance of using string compression methods.

Information compression in a sense is the elimination of repeated information. This elimination could reduce the number of bits needed for illustrating information [5]. In string compression, due to high sensitivity, the “lossless” methods are applied, where unlike the “lossy” methods, the data accuracy is not sacrificed to reach an excellent compression level. Most of the text compression methods are based on statistical or dictionary classes. In dictionary-based class, texts are stored in a data structure named dictionary. Every time a new text fragment is found, it is saved in an entry of the dictionary marked by a pointer; this pointer is used in the process of string compression. The statistical class includes methods that develop statistical models of the text; statistical models can be static or dynamic (adaptive). In general, the modeling stage is set ahead of the coding stage in statistical methods. In the modeling stage, a probability is assigned to the input symbols. In principal, the symbols are coded based on the probability in coding stage [6].

Most statistical models rely on either frequency or context approach. In frequency method, the probability is assigned to a symbol based on the number of times the symbol is repeated. The higher the probability of a symbol, the shorter the code assigned to it. In static model, this probability is a fixed value, but in dynamic model the probability is modified “on the fly” as the input text is being compressed. In the context method, a probability is assigned to a symbol considering its context. This prediction is due to lack of access to the future string. The decoder and the encoder are not aware of the upcoming stream of strings; they only have access to the previously processed symbols, the context of the current symbol [6].

One of the best suggested methods for compression of Unicode strings is applying the lossless Lempel-Ziv-Welch (LZW) method. The LZW algorithm is a member of the Lempel-Ziv family. The Lempel-Ziv technic was introduced in 1977 and it is named LZ77. This algorithm is the basis of “deflate” technic applied in “zip”, which is used in programs like WinZip, PKZip, gzip, and in Unix compression utility. This method keeps its dictionary within the data themselves. For this purpose, the repeated pieces of a string are related to the first observation of the same piece of string using pointers. In 1978, the second version of this algorithm was introduced, named LZ78, where a dictionary is used separately. In 1984 the LZW algorithm was introduced for high-performance disk controllers, but it is mostly used in GIF (Graphics Interchange Format) files. This algorithm is the extended version of LZ78 [7, 8].

One of the Unicode string compression methods is Huffman algorithm [9] introduced in 1952 and has been frequently studied ever since. This algorithm has mostly been used for compression of other languages or as a part of a compression process. In Huffman algorithm, the number of times that each character is repeated in a string is determined. The obtained list is sorted into a descending order and forms a tree, through which an equivalent code is extracted for each character. This code is made of a number of bits and its length may be different for each character. The theory behind this method relies on frequency of some letters versus other letters in natural languages. In this method, the shortest stream of bits is used for the most frequent characters and the longest ones for
the rarest characters. This algorithm is used in some formats like MP3 and JPEG [7, 8, 10].

Applying the Huffman algorithm for compression of Arabic texts yields better results compared with LZW technic, but the opposite holds true for English texts, (i.e. the LZW method outperforms Huffman algorithm in compressing English texts [8]). Applying Huffman algorithm to Arabic strings yields inefficient results in comparison with Persian strings. The Arabic strings are compressed 3.5% less than Persian strings on average. In contrast, short length (e.g. 1024 bytes) Persian strings are compressed less in comparison with Arabic strings with the same length [1].

One of the effective features on the implementation of compression algorithm is the string size. If the size is more than 5 MB, the encountered string is a very big one. In this case, every single word must be considered as a base unit for compression. This compression approach is named word-based [11]. Medium-sized strings are of 200 KB to 5 megabytes. In this case, selecting a word as a base unit for compression is not appropriate; therefore, a smaller grammar part (i.e. syllable) is used. Short strings are from 100 to 200 KB, where every character must be considered as a base unit for compression. Though very small strings are about 160 KB, still the previous compression methods are applied [12, 13]. The size of a SMS is usually less than 140 bytes. Fig.1 shows the size of more than 10100 short messages sent in English [14] and these messages do not contain any spam or advertisements. More than 50% of the messages are smaller than 50 bytes; more than 70% of them are below 75 bytes and only 0.03% of the whole messages have a size of more than 150 bytes.

The size of a message in instant messaging networks like Twitter is way less than 160 bytes. The strings with about 78 characters [15] like the ones sent in instant messaging networks are named “Tiny-strings” by the authors of this study.

Using the compression methods like gzip, bzip2 or PAQ [16, 17] for Tiny-strings is not appropriate. In most cases, the size of overhead data at the beginning of message together with the compressed string is more than the size of non-compressed string [18]. Only SMAZ algorithm [19] appears to be fit for compression of Tiny-strings [18]. This algorithm is presented using a predefined dictionary. The SMAZ algorithm reduces the size of Tiny-strings of SMS by 29% and the Tiny-strings of Twitter by 19% [18]. This algorithm is introduced and evaluated only for non-Unicode strings [18]. Finding a constant text pattern for messages of even one person is not possible; people with an accent for example, apply the common text pattern in their messages for the purpose of concealing their accent, while the same people have their own specific pattern in writing to their family members. The context pattern used in sending a message differs upon the receivers’ social status [20].

With respect to Tiny-strings’ size, the available methods are not efficient enough; therefore, an attempt is made to introduce an efficient compression method for Tiny-strings. This method might be useful in some previous research in big data analysis [21] and data processing [22, 23].

II. THE PROPOSED METHOD

Assume the existence of the character string S as a stream that includes C_s in the form \( S = \{(c, x)|x \in I, c \in char\} \), where \( I = \{1, 2, 3, \ldots, n\} \) indicates the location of the character and \( C = \{1, 2, 3, \ldots, C_{w\text{Max}}\} \) indicates the corresponding character. The goal is to send stream S from point Y to point Z with optimum length. Based on the type of transmitted information, \( C_{w\text{Max}} \) may vary in value (e.g. for sending a stream of characters subject to Unicode standard, every character needs 2 bytes of space and \( C_{w\text{Max}} \) is 65535). Usually for sending any of the \( C_s \), a space equal to the size of \( C_{w\text{Max}} \) is
necessary. The maximum number of bits necessary to express $C_{nMax}$ is shown by $(C_{nMax})_b$ and the size of stream $S$ is equal to $S_{lengthB}$ in bytes, which is calculated through (1):

$$S_{lengthB} = \sum_{k=1}^{n} C_k \times (C_{wMax})_b$$

$C_{support}$ is considered as the number of bits necessary for displaying length of $(C_{nMax})_b$. For example, for sending characters using Unicode, the number of bits necessary to display every $(C_{nMax})_b$ is five as shown in (2):

$$C_{wMax} = 16$$
$$c_{support} = (C_{wMax})_b \text{ Bits required to display } = 5$$

Assume that $C_{LB}$ is the minimum number of bits necessary to display the length of every $C_i$ in bits, with a range in accordance to (3):

$$1 \leq C_{LB} \leq c_{support}$$

A. First stage of compression

At the beginning, every new $C_i$ available in stream $S$ is fed into a table named the mapping table which has only 2 columns. The first and second columns contain the mapping value and the new $C$, respectively, as shown in Table I. The mapping value starts from one and increases by one unit in each row. The character string $S$ is checked from beginning to end and every time a character is observed for the first time it is added to this table. If the new characters are expressed by $C_n$ and the existing ones by $C$, the $S$ can be expressed as $S = [C_n, C_n, \ldots, C_n]$. Results in four different streams expressed in (4):

$$S = [C_n, C_n, \ldots, C_n]$$
$$S_{LB} = [C_{LB}, C_{LB}, \ldots, C_{LB}]$$
$$S_{Index} = [Null, Null, \ldots, C_{Index}]$$
$$S_{Index-LB} = [Null, Null, \ldots, C_{Index-LB}]$$

Where, $S_{LB}$ corresponds to $S$, provided that $C_{LB}$ is the length of $C_i$ in bits. $S_{Index-LB}$ correspond to $S_{Index}$, provided that $C_{Index-LB}$ is the length of $C_{Index}$ in bits (and for $C_{Ini}$, the $C_{Index}$ and $C_{Index-LB}$ values are set to Null in the first observation).

B. Second stage of compression

Based on a specific policy, it is possible to break the $S$ string into smaller strings. The borderline between the new and repeated characters is recognized as the potential breaking points.

The letter $A$ is the first potential breaking point, as shown in Fig. 2.

The potential breaking points are expressed as $S = S_{pn1} \cap S_{pn2}$, $S_{pn3} \cap S_{pn4}$, \ldots. The value of $C_{New-Count}$ is the number of consecutive new characters in $S_{LB}$ stream and $C_{Index-Count}$ is the number of consecutive mappings in $S_{Index-LB}$ stream. Fig. 2 shows how these two values are calculated.

For drawing a conclusion regarding the potential breaking point(s) of $A$, (5) is applied.

In case $f_1(S_{pn1}, S_{pn2}) \leq f_2(S_{pn1}, S_{pn2})$, the potential breaking point of $A$ is not taken into account and two sections of $C_n$ and $C_i$ are considered as $C_n$. Here, the corresponding value of $S_{pn2}$ is considered as Null in $S_{Index}$ and $S_{Index-LB}$ streams; but if $f_1(S_{pn1}, S_{pn2}) > f_2(S_{pn1}, S_{pn2})$, the potential breaking point of $A$ is considered in stream $S$. If there is no breakage between these two regions, $S_{pn1}$, $S_{pn2}$, and $S_{pn3}$ form a bigger $S_{pn1}$, the decision making process is carried out for other potential breaking points to the end of stream $S$.

<table>
<thead>
<tr>
<th>TABLE I</th>
<th>MAPPING VALUES TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping value</td>
<td>$C_n$ value</td>
</tr>
<tr>
<td>1</td>
<td>$C_n$</td>
</tr>
<tr>
<td>2</td>
<td>$C_n$</td>
</tr>
<tr>
<td>3</td>
<td>$C_n$</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
</tr>
</tbody>
</table>

If $C_{Index}$ is considered as the mapping value of $C_{ui}$ after first observation, $C_{Index-LB}$ can be expressed as the number of bits necessary for expressing the length of $C_{Index}$. For the $C_{ui}$S, the $C_{Index}$ value is considered Null in the first observation, that
\[ f_1(S_{pn1}, S_{pn2}) = (C_{nLBMax} \times (C_{Index-Cont} + C_{New-Cont} + 1)) + C_{support} + 1 \]

\[ f_2(S_{pn1}, S_{pn2}) = (C_{nLBMax} \times (C_{New-Cont} + 1)) + (C_{Index-LMax} \times (C_{Index-Cont} + 1)) + 2(C_{support} + 1) \]

C. Third stage of Compression

At this stage, \( S \) is divided into smaller parts using appropriate breaking points. For more simplicity, only string \( S_{LB-pn} \) is considered, which corresponds to \( S_{pn} \) region in \( S_{LB} \) stream. At first, a maximum variable with \( -\infty \) value is set and the points in which there is a change in maximum value are determined from the beginning of \( S_{pn} \) stream. The manner in which the maximum values change is given in (6).

\[ S_{LB-pn} = \{C_{nLBMax} + 1, C_{nLB}, \ldots, C_{nLBMax}, C_{nLB}, \ldots, C_{nLBMax} \} \]

(6)

The value of \( Count \) is the number of \( C_n \)s starting from \( C_{nLBMax} \) up to the one before \( C_{nLBMax+1} \) and the value of \( K \) is the possible number of consecutive \( C_{nLBMax+1} \); \( K \) is at least equal to one.

Calculation of these two values are shown in Fig. 3.

\[ S_{pn} = \{C_n, \ldots, C_n, C_n, \ldots, C_n, \ldots, C_n\} \]

\[ S_{LB-pn} = \{C_{nLBMax} + 1, C_{nLB}, \ldots, C_{nLBMax}, C_{nLB}, \ldots, C_{nLBMax} \} \]

Fig. 3 Calculating the values of \( Count \) and \( K \) for \( Part_0 \)

The first appropriate breaking point is symbolized by \( P_0 \) in Fig. 3. This point can be assessed in \( S_{pn} \) stream as an appropriate breaking point.

\[ Part_0 = (C_{nLBMax+1} \times (k + 1 + Count)) + C_{Max} + 1 \]

\[ - (C_{nLBMax} \times (Count + 1)) + 2(C_{support} + 1) + Sum \]

(7)

In (7), before making the final decision for the first break, the value of \( Sum \) is considered as zero. The manner of assigning a value to \( Sum \) will be discussed in due course. The same process is run for the points after \( C_{nLBMax+1} \) and \( C_{nLBMax+2} \), while the previous region(s) are considered in calculation of \( Count \). The manner of calculating \( Count \) and \( K \) for \( Part_1 \) is presented in Fig. 4. Equation 8 evaluates \( Part_1 \) for breaking from the appropriate point \( P_1 \). By expanding the same method, the \( C_n \)s can be assessed up to \( C_{nLBMax} \). The largest positive value among the \( h-1 \) calculated values is selected and is named \( Part_{best} \). This process is shown in (9).

\[ S_{pn} = \{C_n, \ldots, C_n, C_n, \ldots, C_n, \ldots, C_n\} \]

\[ S_{LB-pn} = \{C_{nLBMax+1}, C_{nLBMax+2}, \ldots, C_{nLBMax}, C_{nLB}, \ldots, C_{nLBMax} \} \]

\[ P_1 \]

Fig. 4 Calculating the values of \( Count \) and \( K \) for \( Part_1 \)

\[ Part_1 = ((C_{nLBMax+3} \times (k + 1 + Count)) + C_{Max} + 1) \]

\[ - ((C_{nLBMax+2} \times (Count + 1)) + 2(C_{support} + 1) + Sum) \]

(8)

\[ Part_{best} = \text{Max}(Part_i) > 0 \forall i = 0, \ldots, h - 1 \]

(9)

The stream \( S_{LB-pn} \) breaks at the point \( C_{nLBMax} \) which is related to \( part_{best} \) and the stream \( S_{pn} \) breaks at the corresponding point of \( part_{best} \). The manner of this breakage and the two small regions named \( S_{small1} \), \( S_{small2} \) are illustrated in Fig. (5). The value of \( Sum \) is changed according to (10).

\[ Sum = Part_{best} - C_{nLBMax} \]

If all the calculated values for parts are less than zero, only the second point which is related to \( C_{nLBMax} \) is considered, Fig. 6.

\[ S_{pn} = \{C_n, \ldots, C_n, C_n, \ldots, C_n, \ldots, C_n\} \]

\[ S_{LB-pn} = \{C_{nLBMax+1}, C_{nLBMax+2}, \ldots, C_{nLBMax}, C_{nLB}, \ldots, C_{nLBMax} \} \]

\[ Part_{best} \]

Fig. 5 \( S_{small1} \) and \( S_{small2} \) regions

\[ Sum = Part_{best} - C_{nLBMax} \]

(10)

Now starting from the first \( C_{nLB} \) value after \( C_{nLBMax} \) in \( S_{LB-pn} \) stream and by considering the maximum as \( -\infty \), the maximums are searched for and the whole third step is repeated. Here, if all the calculated values for parts are less than zero, only the second point is considered related to \( C_{nLBMax} \) and a new region is formed together with the previous region, Fig. 7.
As observed in Fig. 9, the first bit is one. The value of \( C_{\text{support}} \) for the next bit is set equal to the largest length of mappings available in \( S_{\text{Small}} \) in the binary system. Next, every \( C_{\text{Index}} \) in \( S_{\text{Small}} \) is set at the end of bits with a fixed size equal to the largest length of \( C_{\text{Index}} \) available in \( S_{\text{Small}} \). Finally, a \( \text{Null} \) value is added to \( S_{\text{Small}} \) in the binary system with the size of the largest length of mappings available in \( S_{\text{Small}} \).

The same process is run for other \( S_{\text{Small}} \) and \( S_{\text{Small}} \) available in stream \( S \). At the end, all the bits form a queue and the resultant bit stream is changed to bytes. These bytes are sent to destination \( Z \) as a compressed string of \( S \). In this process, the number of bits should be a multiple of 8, otherwise a number of zero(s) is/are added to the last byte to fill the space of the absent bits; the zero(s) should be inserted in a manner where the value of the last byte is not lost.

**E. The receiver**

The stream \( B \) in (12) is the same received bytes from source \( Y \).

\[
B = [b_{11}, b_{12}, b_{13}, b_{14}, b_{15}, b_{16}, b_{17}, b_{18}, b_{19}, b_{20}, b_{21}, b_{22}, b_{23}, b_{24}, b_{25}, \ldots]
\]

At this point, a mapping table is considered similar to the one available in sender, where there is no value initially. If \( b_{11} \) is zero, the value of the next \( C_{\text{support}} \) bits is known as the \( \text{Length} \) of this region (The \( C_{\text{support}} \) value is known by the sender and receiver as a detail of the data type). Next, the reading of \( C_{n} \) begins from the bit \( C_{\text{support}} + 1 \) with \( \text{Length} \) of the same region. Every new \( C_{n} \) absent in the mapping table is added to the table. This process is repeated until a \( \text{Null} \) value is obtained. Then, another bit is read, if its value is equal to zero, the same process is repeated using the new value of \( \text{Length} \).

If the value of the first bit in the region is one, the previous process is repeated, but the \( C_{\text{Index}} \) value is read instead of \( C_{n} \) that is, it was once inserted in the mapping table and is repeated. Thus by referring to the mapping table, the equivalent \( C_{n} \) of the \( C_{\text{Index}} \) is read. The values read directly or indirectly (from the mapping table) form the stream \( S \) to be sent.
III. EXPERIMENTAL RESULTS

One of the common challenges in assessing the text compression methods is lack of related data sets. In this study, a set of tweets (Tiny-strings) in Arabic language is used, which contains 2000 Tiny-strings of modern standard Arabic with Jordanian dialect. To transmit these tweets, applying Unicode is mandatory. The size of the tweets are shown in Fig. 10 [24]. About 50% of the tweets are less than 100 bytes; only 31% of them have a size above 160 bytes.

![Fig. 10. The size of the Arabic tweets and their number](image1)

To assess the proposed algorithm, the Huffman algorithm, known as the best compression method [8, 25], is applied here. These assessments are run on a laptop with an Intel(R) Core(TM) i3 CPU M 330 @ 2.13GHz processor. The percentage of size reduction in Arabic strings using the proposed algorithm and Huffman method is compared in Fig. 11. The newly proposed algorithm outperforms the Huffman algorithm by 12% in reducing the size of Tiny-strings of Arabic language. The run time of the two algorithms is compared in Fig. 12. This proposed algorithm is 0.26 seconds slower than Huffman algorithm.

![Fig. 11. Comparison of this proposed algorithm and Huffman method in size reduction of Arabic Tiny-strings](image2)

For assessing this proposed algorithm for compression of Persian Tiny-strings, a data set containing more than 22400 tweets obtained from Persian and international news broadcasting corporations is used [26]. These tweets are selected equally with no bias from the most recent posts in Twitter; it must be noted that applying Unicode is mandatory here. Fig. 13 shows the size of the tweets.

![Fig. 12. Comparison of execution time of this proposed algorithm and Huffman method for Arabic Tiny-strings](image3)

The percentage of size reduction of Persian tweets is compared for proposed algorithm with that of the Huffman method in Fig. 14. This proposed algorithm outperforms Huffman algorithm by more than 8% in reducing the size of Tiny-strings in Persian language. The execution time of the two methods is compared in Fig. 15. The findings on both Persian and Arabic languages are tabulated in Table II.

![Fig. 13. The size of Persian tweets and their number](image4)

The percentage of size reduction of Persian tweets is compared for proposed algorithm with that of the Huffman method in Fig. 14. This proposed algorithm outperforms Huffman algorithm by more than 8% in reducing the size of Tiny-strings in Persian language. The execution time of the two methods is compared in Fig. 15. The findings on both Persian and Arabic languages are tabulated in Table II.

![Fig. 14. Comparison of size reduction of Persian Tiny-strings through proposed algorithm and Huffman method](image5)

![Fig. 15. Comparison of execution time of proposed algorithm and Huffman method for Persian Tiny-strings](image6)
IV. CONCLUSION

An appropriate compression method is proposed here for Unicode Tiny-strings (which are about 78 characters long). This method is fit for compression of short messages and strings sent in instant messaging networks or any similar application. This method is subject to no restriction in the language or the number of languages available in the text. The results obtained from the assessment on Arabic and Persian Tiny-strings indicate that this method outperforms the Huffman algorithm in compression of the strings. Regarding compression of Tiny-strings, applying this proposed algorithm is more efficient for Arabic than Persian language.

For future studies, assessing the proposed method for compression of other languages or other types of data streams [27] would be advantageous. The algorithm can be compared with other proposed methods [28, 29] and the compatibility of the method can be checked for compression of webpages [30] especially on mobile phones.

ACKNOWLEDGMENT

This research is supported by Petro Palayesh Kian Espadana Company. Also the authors would like to greatly thank Dr. Ahmad Baraani-Dastjerdi, Dean of Computer Engineering Faculty, University of Isfahan, for his invaluable help and comments. Without his persuasion and cooperation, this study would not have begun.

REFERENCES


Abbas Malekpour is currently an Assistant Professor in the Institute of Distributed High Performance Computing at University of Rostock. He received his Master Degree from Stuttgart University and his Ph.D. degree from University of Rostock, Germany. From 2002 to 2004 he was with Institute of Telematics Research Group at university of Karlsruhe, Germany. And from 2004 to 2010, he was a research assistant in MICON Electronics and Telecommunications Research Institute at University of Rostock, Germany. His current research interests include the areas of Mobile and Concurrent Multi-path Communication prototyping.

Email: abbas.malekpour@uni-rostock.de

Peter Luksch received his Ph.D. degree in Parallel Discrete Event Simulation on Distributed Memory Multiprocessors from Technische Universität München, Germany.

He finished his Postdoctoral Lecture Qualification (Habilitation) in Increased Productivity in Computational Prototyping with the Help of Parallel and Distributed Computing. Currently, Prof. Dr. rer. nat. habil Peter Luksch is head of the department of Distributed High Performance Computing at University of Rostock, Germany

Email: Peter.luksch@uni-rostock.de

Mohammad Reza Mojtabaei was born in 1967, in Isfahan, Iran. He got his Master degree in Machine Intelligence & Robotic, from Shiraz University, Shiraz, Iran, in 1995. He is currently a PhD candidate at University of Isfahan, Isfahan, Iran. He is also working as an Instructor at computer group in technical & vocational university, Mohajer college, Isfahan, Iran. His research interests include pattern recognition, image processing, programming languages, data structure and algorithm design.