International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:7, 2007

Database Compression for Intelligent On-board Vehicle Controllers

Ágoston Winkler, Sándor Juhász, and Zoltán Benedek

***Abstract*—**The vehicle fleet of public transportation companies is often equipped with intelligent on-board passenger information systems. A frequently used but time and labor-intensive way for keeping the on-board controllers up-to-date is the manual update using different memory cards (e.g. flash cards) or portable computers. This paper describes a compression algorithm that enables data transmission using low bandwidth wireless radio networks (e.g. GPRS) by minimizing the amount of data traffic. In typical cases it reaches a compression rate of an order of magnitude better than that of the general purpose compressors. Compressed data can be easily expanded by the low-performance controllers, too.

***Keywords*—**Data analysis, data compression, differential encoding, run-length encoding, vehicle control.

I. Introduction

To operate a public transportation network successfully, it is essential to provide the passengers with the appropriate traveling information. There is a wide variety of equipments supporting these functions including graphical signs, voice announciators, positioning and communication devices [1]. An important component of on-board systems is the controller that coordinates the work of all the other devices (Fig. 1).

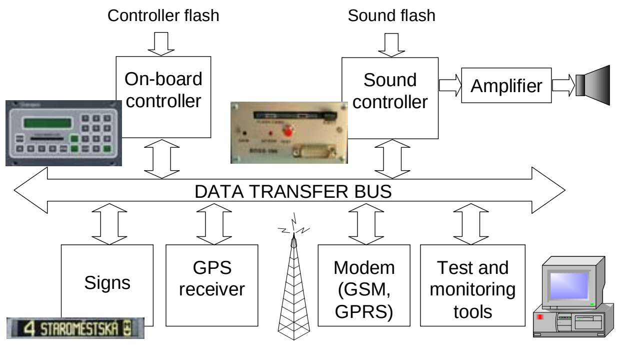

Fig. 1 Structure of on-board systems

The on-board controllers should always hold the most current data for their optimal operation. A simple and frequently used way for providing this is the manual update using different memory cards (e.g. *flash cards*) or portable computers, however this method takes a lot of time and requires human contribution. It would be much more comfortable to transmit the new database to the controllers using wireless radio networks. Unfortunately many transportation companies cannot afford the operation of high bandwidth radio networks (e.g. WLAN). In addition, it is an obvious expectation that older controllers should be used as well, which may not be able to be connected to such a network. Typical controller data files are however too large to be transmitted over a low bandwidth connection as most of the vehicles leave the garage and arrive back at the same time, leaving only a short period of time for updating almost every controller. Different vehicles may require different databases so broadcast techniques cannot be used.

A solution for this problem is the compression of the databases which is promising for more reasons. Firstly, data files have a regular structure, they contain many repetitions. Furthermore, in typical cases only a few modifications are made in the database so only small segments of the files are affected. This means that differential encoding can be applied successfully if the controller has enough storage capacity. In the examined cases the latter condition was fulfilled.

It must be noticed that most on-board controllers have a relatively simple processor. This implies that the decoding algorithm should be as simple as it is possible so that controllers can expand the files in real time. On the contrary, encoding might be more resource-intensive as it is executed by a computer with a powerful processor and having a less strict time limit for this operation.

This paper describes a compression algorithm specialized for this problem. It summarizes the basic techniques of data compression, then demonstrates how the logical and physical properties of the data files were examined and how the suitable compression techniques and their parameters were chosen, in order to reach a better result than general purpose compressors do. Finally it presents the steps of the encoding process and reports the achieved results.

II. Related Works

A. Terminology

Data compression requires two procedures: *compression (encoding)* transforms the original data into *compressed data*, whereas *decompression (decoding)* restores it (Fig. 2).

Manuscript received July 15, 2005.

The authors are with the Department of Automation and Applied Informatics, Budapest University of Technology and Economics, 1117 Budapest, Magyar tudósok körútja 2., Hungary (e-mail: awinkler@t-email.hu, sanyo@aut.bme.hu, benedek.zoltan@aut.bme.hu).

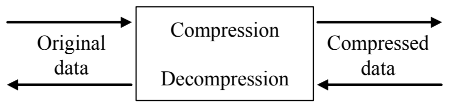

Fig. 2 Basic scheme of data compression

The efficiency of the compression can be measured by the *compression rate* which is defined as (1). The higher value of the compression rate indicates a better compression [2].

$$\textit{Compression rate} = \frac{\textit{Original data size}}{\textit{Compressed data size}} \qquad (1)$$

B. *Classification*

There are two basic types of compression techniques: *lossy and lossless methods*. Lossy compression can be much more effective for certain data types than lossless but the original data cannot be restored with full accuracy. In some cases it is acceptable (e.g. picture and sound compression), however the particular problem does not allow any losses: the contents of controller data files must be restored absolutely precisely.

An other classification distinguishes logical and physical compression techniques [2].

Logical compression uses and may modify the structure of the files, such as normalizing databases or changing data storage formats. Although controller files have a regular structure, their exact format should not be exploited as even small changes brought by future versions would make the algorithm unusable so both the encoding and the decoding program would have to be modified and reloaded into the controllers. This implies that some properties of the files derived from their structure should be utilized, however the meaning of each bit should not be exploited, in order to create a robust algorithm that is not sensitive to smaller changes in the format.

Physical compression uses general statistics so it can be applied for all file formats. The efficiency of physical methods can be increased by examining which of them can be used with the best results when compressing files of a specific format, and optimizing the encoding scheme based on the results.

According to the considerations above, physical lossless techniques should be used particularly. The next part of this chapter summarizes the most general methods of this type [2].

C. *Physical Lossless Techniques*

Differential encoding is a very effective way of data compression if the previous version of the file is available during the decompression and there are relatively few changes.

Run-length encoding (RLE) is worth using when identical characters appear together in long series [3]. In this case it is enough to identify the specified character and store the length of the series.

An other method may be considered as an *enhanced version of RLE*: it encodes character strings by specifying their length just as RLE does but characters do not have to be identical. Instead of a character, a distance is specified: the characters can be taken from a preceding part of the file (that has been already decompressed) starting from the actual position minus the given distance. This is the same technique as the one used by differential encoding but repetitions are searched for within the file so it can be used even when the previous version of the file is inaccessible. However, in general only shorter matches are found with this method.

Bit-mapping can be used if a certain character is very frequent but does not form long series. In this case a “map” identifies the positions of the frequent character (e.g. one byte describes the following block of 8 characters) and only the other characters have to be specified [2].

Pattern substitution is a technique that takes advantage of character strings having a special function in certain file types. Depending on the length of these strings a good compression rate can be achieved by replacing them with a shorter code (e.g. one simple character) [4].

Half-byte packing tries to put two characters in one single byte. If data files (or some parts of them) contain only a few character values (e.g. numbers and operators in business data files) 4 bits may be enough [2].

Diatomic encoding does not encode single characters but character pairs [2].

Statistical encoding uses different code lengths for characters with different probabilities of occurrence. Statistical methods are very efficient (the *Huffman code* is optimal) but their decoding requires too many bit operations causing that the decompression may get too complex for the low-performance on-board controllers [5].

Dictionary-based techniques (e.g. the *Lempel–Ziv–Welch method*) reach very good compression rates as well and they are extremely fast. However they use typically 12-15 bit long words causing the same performance problem with the decompression [6].

III. Analysis of the Data Files

A. *Logical Structure*

The database contains all the information required for informing the passengers about the route of the vehicle. The logical structure of data files used by different on-board systems is quite similar, the data is usually stored in a tree. Fig. 3 shows the structure of *Vultron* controller data files. As it was mentioned earlier, this structure may help to determine the parameters of some physical compression methods so it is worth examining.

A typical controller data file contains the name, the author and the creation date of the file, the type of the equipments the database relates to (controller, LCD sign, GPS, odometer, keyboard etc.) and the text of the messages for the operator. It may also contain a table of the character sets and configurations stored in the file and the sounds stored in a separate sound flash. A configuration keeps information about a specific vehicle or vehicle type, including the type of the components of the on-board system (e.g. stop and route related signs, odometer's impulse density etc.) and the description of the routes the vehicle runs on. A route is a series of stops with some own properties, e.g. public and internal route codes, type, picture id of the route related signs, in addition to the stop specific information. Stops have properties like name, GPS coordinates, picture id of the stop related signs, sound id of the announcements.

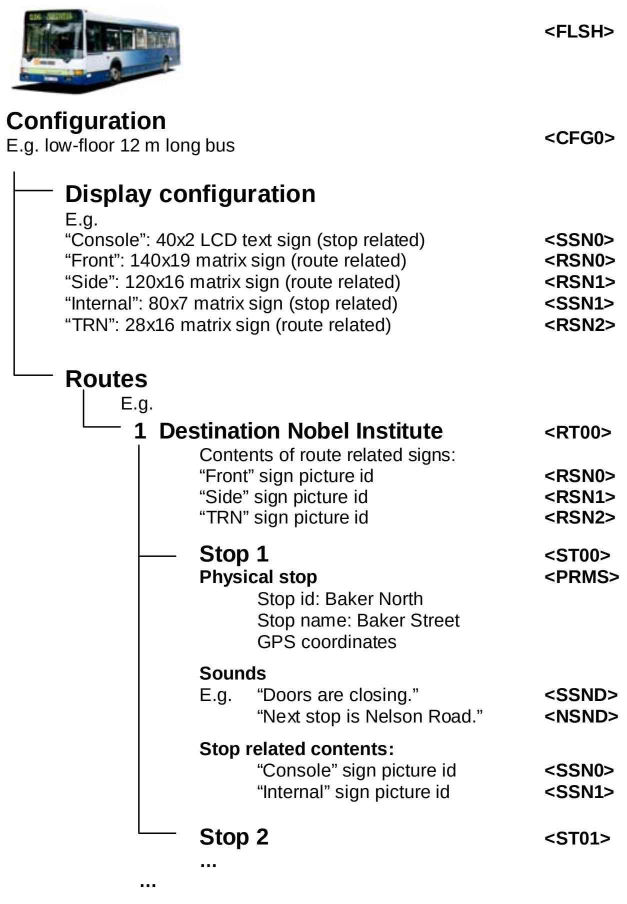
<FLSH>
Configuration
E.g. low-floor 12 m long bus
<CFG0>
Display configuration
E.g.
"Console": 40x2 LCD text sign (stop related) <SSN0>
"Front": 140x19 matrix sign (route related) <RSN0>
"Side": 120x16 matrix sign (route related) <RSN1>
"Internal": 80x7 matrix sign (stop related) <SSN1>
"TRN": 28x16 matrix sign (route related) <RSN2>
Routes
E.g.
1 Destination Nobel Institute <RT00>
Contents of route related signs:
"Front" sign picture id <RSN0>
"Side" sign picture id <RSN1>
"TRN" sign picture id <RSN2>
Stop 1 <ST00>
Physical stop <PRMS>
Stop id: Baker North
Stop name: Baker Street
GPS coordinates
Sounds
E.g. "Doors are closing." <SSND>
"Next stop is Nelson Road." <NSND>
Stop related contents:
"Console" sign picture id <SSN0>
"Internal" sign picture id <SSN1>
Stop 2 <ST01>
...
...

Fig. 3 Structure of Vultron controller data files

B. *Physical Properties*

To determine which compression techniques are worth using, physical properties of the data files must be analyzed. As on-board systems are used by different companies for more or less different purposes, more test files have been examined. The test file "BKV" is used by the local transportation company of Budapest whereas "Volán" comes from a regional bus company of Hungary. The main difference between the two files is that "Volán" contains only route related sign pictures whereas "BKV" stores stops with their sign pictures and sound references as well. Two files from Riga ("Imanta" and "Talava") used by the local transportation companies have been tested, too.

In the first test *the frequency of the characters* were examined. (Test files use 8 bit encoding.) Table I contains the 8 most frequent characters in each test file. As it can be seen the distribution of the frequencies is not uniform: the first and the second most frequent characters are much more frequent than any others.

TABLE I
THE EIGHT MOST FREQUENT CHARACTERS IN THE TEST FILES

	BKV		Volán		Imanta		Talava	
1	Null	34.6%	Null	32.9%	Null	44.3%	Null	46.6%
2	Space	12.1%	Space	9.3%	Space	7.9%	Space	6.6%
3	S	3.3%	R	2.3%	S	4.0%	S	4.6%
4	<1>	2.5%	1	2.1%	<1>	2.9%	<1>	3.0%
5	N	2.0%	S	2.0%	<2>	1.9%	N	2.0%
6	<2>	1.7%	N	1.9%	N	1.8%	<2>	2.0%
7	0	1.6%	c	1.9%	0	1.4%	a	1.4%
8	<12>	1.4%	<1>	1.6%	a	1.2%	0	1.1%

<N> means the character of the value n.

The distribution of the relatively rare characters is not uniform either. These results suggest that characters should not be encoded by using 8 bits by all means. Fig. 4 shows that a less number of bits would be enough for the major part of them.

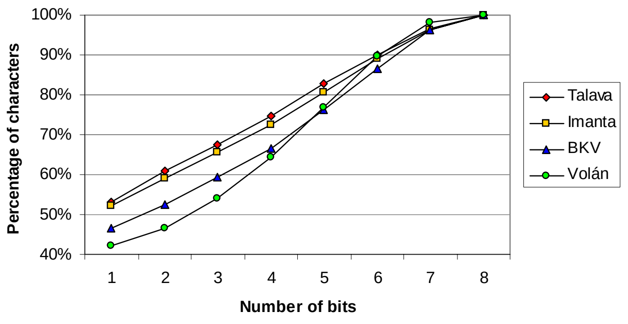

Fig. 4 Percentage of the characters that can be encoded by using the given number of bits, in different test files

To check if RLE can be employed, *the length of homogeneous character series* should be examined. As the aim is to encode them binary, it was examined how many percent of them can be described by using a given number of bits. Fig. 5 shows the results.

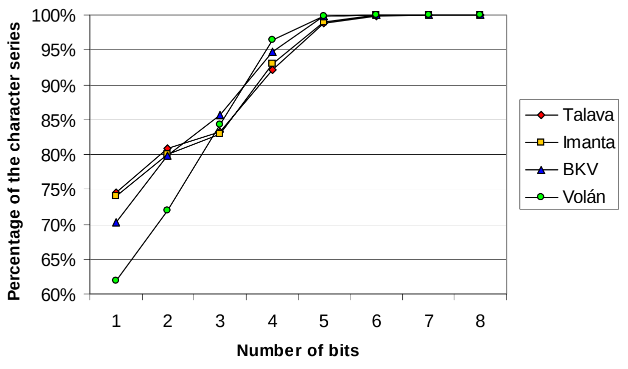

Fig. 5 Percentage of the homogeneous character series that can be encoded by using the given number of bits, in different test files

It can be seen that there are quite long series as well so this technique is worth using. The longest series are formed by the most frequent characters: more than 99% of the homogeneous series longer than five characters contain nulls or spaces.

In order to examine the efficiency of *pattern substitution*, the test files were searched for such strings and it was found that the most frequent ones are the starting segments of the structure headers. As these strings characterize the file format itself, the compression algorithm may use a fixed code for them. There were some other frequent strings found, however different ones in different test files. These location and company dependant strings should not be built into the algorithm because it has to be effective in all environments. Nevertheless enhanced RLE makes it possible to utilize them in the compression, too.

To find out how far it is worth seeking for *enhanced RLE*, an other test was made. Fig. 6 shows the average length of the longest repetitions found by using a specified maximal seeking distance, starting from different positions. As all test files gave similar results, only the graph based on the examination of "Volán" is presented.

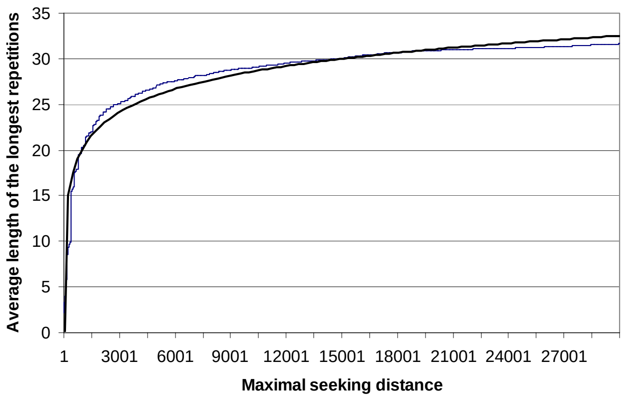

Fig. 6 The average length of the longest repetitions found by using the specified maximal seeking distance in the test file "Volán", with the trend line

The graph is similar to the trace of the logarithm function: it raises steeply until the distance of about 2000 characters, then it flattens significantly. This is the point where the maximal seeking distance should be determined: it is not worth searching from further positions because the benefit would be much smaller than the time required by the wider search.

IV. Algorithm and Parameters

Based on the results of the physical analysis of the data files it was decided to use RLE for null and space characters, pattern substitution for the most frequent structure headers, moreover enhanced RLE and differential encoding. If more of these techniques can be used at a specified position of the file, the compression algorithm always chooses the one that assures the greatest compression benefit. This behaviour is *greedy* but it is acceptable as the following example shows. If there is a long series that could be described in one step using e.g. RLE, it is possible that some characters at the beginning of the series are encoded together with the characters preceding them, using an other compression technique. However the algorithm will apply RLE for the rest of the series so if there is a waste at all, it will be only a few bytes. On the other hand, this simplification results a great improvement in compression time.

The next step is to determine *the searching parameters* for enhanced RLE and differential encoding: the maximal seeking distance and the maximal length of the repetitions. The greater these values are the better compression rate one can reach (using a suitable representation) but also the more time the encoding process will take. So a compromise must be made between these two factors. As compression time is not so important in this case, it was decided to use the following factor of quality (2).

$$\text{Factor of quality} = \frac{\text{Compression rate}^2 \cdot \text{Original size}}{\text{Compression time}} \qquad (2)$$

The higher priority of the compression rate is emphasized by the second power. To determine the searched values, an iterative technique was applied. Firstly, an initial encoding scheme was created based on the earlier results, then the compression algorithm was tested by using each combination of the maximal length and the seeking distance. Both parameters were chosen between 1 and 15 as the power of 2: this meant 225 test cases. The final encoding scheme was determined based on the new results.

During the test, the compression rate, the compression time and the factor of quality were measured. In order to test differential encoding as well, modified versions of the test files "BKV" and "Volán" were created: some new stops and lines were added to "BKV" but nothing was erased from this file. On the contrary, some records from "Volán" were removed: this proved to be a significant change as the id of all the elements following the deleted records changed. These records are however referenced in other parts of the file so long blocks cannot be taken from the old version after this modification.

Differential encoding arose an other problem: keeping the old and the new file synchronized. If too many records are added or removed from the database, the seeking distance may get too low. However it would be enough if the starting position were adapted to the changes detected in the file. Unfortunately this causes an other problem: the string at the actual position can be found in the old version accidentally, too, particularly shorter ones. It is quite difficult to decide if such a match gives reason for the synchronization or not. If the wrong decision is made frequently, the result will be similar to the one experienced when using no synchronization at all: synchronicity may get lost. To solve this problem, two starting positions were used, a standard and an adapted one. The standard position equals always the actual position of the new file whereas the adapted one is updated every time when differential encoding is used: it is set to the character following the encoded string in the old file. The searching process is always done from both positions: if their intervals have a common part, the concerned positions are examined

International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:7, 2007

only once. If the adapted position goes wrong it may be restored by using the standard position. Tests showed that this occurs rarely but sometimes it is really necessary to make the algorithm work properly.

Fig. 7 shows the results of the compression test of the file “BKV”. Other test cases representing typical use cases delivered quite similar results. The conclusion was unambiguous: the maximal seeking distance should be $2^{11} = 2048$, whereas the maximal length of repetitions to be encoded should be $2^{15} = 32768$.

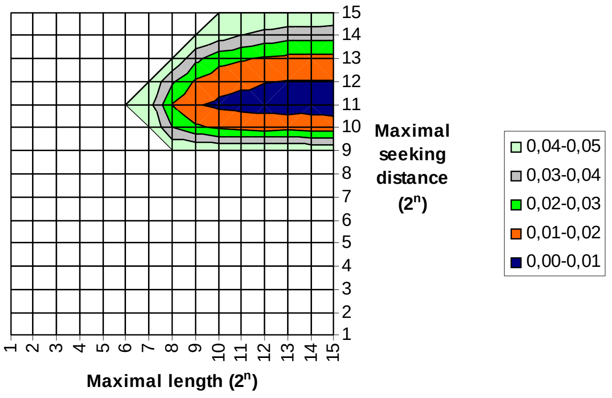

Fig. 7 Reciprocal of the factor of quality by using different parameters (2^n) in the test file “BKV”

The previous test was actually only a simulation as the optimal encoding scheme was not known. Therefore it was assumed that every length–distance combination could be encoded by using an optimal division of the encoding bytes. However only a few fixed divisions can be used during the real encoding. An other purpose of the simulation was to determine these divisions so that most of the compression commands can be encoded really by using the fewest number of bytes.

As already mentioned, the third aim of the test was to determine the final encoding scheme. The basic principles were the following: more frequent compression methods should be described in a shorter way, the number of bit operations should be minimized and there should be no unused bits. Fig. 8 shows the final encoding scheme.

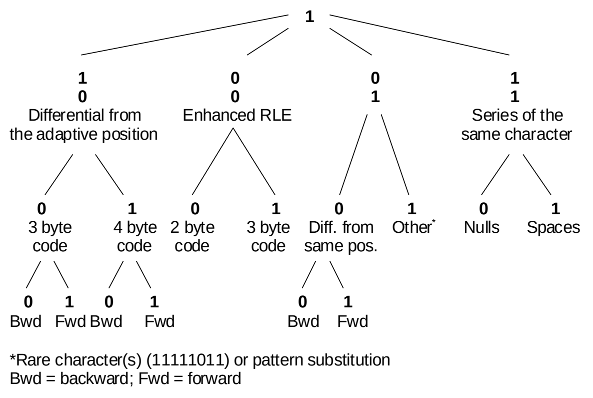
1
1 0 Differential from the adaptive position
0 0 Enhanced RLE
0 1
1 1 Series of the same character
0 3 byte code
1 4 byte code
0 2 byte code
1 3 byte code
0 Diff. from same pos.
1 Other*
0 Nulls
1 Spaces
0 Bwd
1 Fwd
0 Bwd
1 Fwd
0 Bwd
1 Fwd
*Rare character(s) (11111011) or pattern substitution
Bwd = backward; Fwd = forward

Fig. 8 The final encoding scheme

As 96-98% of the characters can be described by using 7 bits (Fig. 4), the least significant bit is used to distinguish command and character bytes. Character bytes represent exactly one character that can be looked up in a table that is stored at the beginning of the compressed file. Command bytes describe compression methods and may be followed by other bytes: the number of the following bytes depends on the type of the compression that is stored by the less significant bits of the command byte. “Rare” characters can be encoded by using a special command byte, too, that is followed by the code of the rare character(s). (The most significant bit indicates if there are more rare characters following.) Fig. 9 shows the share of the different compressing methods in the number of applications. Differential encoding is applied not so often like enhanced RLE, however this method provides 97% of the compression benefit. When the previous version of the file is not used, enhanced RLE has the greatest share in this respect, too.

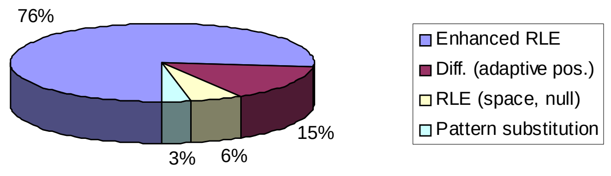

Fig. 9 Share of the different compressing methods in the number of applications in the test file “BKV”

Differential encoding from the standard position is very rare so it was decided that this method is used only for synchronization. It means that only the distance is encoded, the repetition itself (i.e. the length) is described by the next compression command (in most cases using differential encoding from the adaptive position).

Enhanced RLE can be represented by 2 and 3 bytes as well so that shorter and nearer repetitions can be encoded more compactly. Similarly, differential encoding from the adaptive position can be described by using whether 3 or 4 bytes.

V. Results

The compression and the decompression algorithms have been implemented (later referred as *VasComp*) and compared with other, general purpose compressors. Table II shows the results of the comparison.

TABLE II
COMPRESSION RATES USING DIFFERENT COMPRESSORS

Old version	New version	Vas Comp	PKZIP	ARJ	RAR	Vas-Comp + ZIP
BKV	BKV-M	59.80	3.03	3.04	3.23	66.67
BKV-M	BKV	263.11	3.03	3.06	3.24	**261.75**
Volán	Volán-M	19.74	4.57	4.68	5.11	30.94
Volán-M	Volán	19.85	4.58	4.69	5.12	31.28
BKV	Volán	3.27	4.58	4.69	5.12	4.56
Volán	BKV	2.47	3.03	3.06	3.24	3.20
–	Volán	3.24	4.58	4.69	5.12	4.52
–	Talava	3.07	4.44	4.45	5.00	4.33
–	Imanta	2.95	4.01	4.05	4.43	3.90
–	BKV	2.38	3.03	3.06	3.24	3.08

"X-M" means the modified version of the test file "X".

As it can be seen, in typical use cases VasComp reached a compression rate of at least an order of magnitude better than that of the general purpose compressors. In the second case *PKZIP* could not do any further compression on the file. On the other hand, VasComp performed in an acceptable way in non-typical cases as well (cases 5-10: "old version" does not exist or is not really a previous version of the "new file").

Table III contains the compression times of VasComp.

TABLE III
FILE SIZES (BYTES) AND COMPRESSION TIMES (MILLISECONDS)

Old version	New version	Original file size	Vas-Comp time	PKZIP time	ARJ time	RAR time
BKV	BKV-M	157750	375	46	63	66
BKV-M	BKV	152602	78	40	60	63
Volán	Volán-M	714826	2281	120	176	400
Volán-M	Volán	714412	2234	120	176	396
BKV	Volán	714412	19625	120	176	396
Volán	BKV	152602	9890	40	60	63
–	Volán	714412	6718	120	176	396
–	Talava	274114	2796	50	80	96
–	Imanta	271564	2921	67	86	106
–	BKV	152602	2078	40	60	63

"X-M" means the modified version of the test file "X".
Tests were executed by a PC running Microsoft Windows XP 5.1 SP2, with an AMD Athlon XP 1800+ 1.53 GHz processor, 512 MB RAM and an IBM 120 GB 7200 rpm hard disk with 8 MB cache.

It must be admitted that in this respect general purpose compressors proved to be better. However the times of VasComp are acceptable as well and the environment where it was intended to be used does not require extra short compression times. On the other hand, decompression is quite simple and can be done very quickly even by simple processors as it is needed in the particular environment.

VI. CONCLUSION

This paper presented an algorithm that enables the transmission of intelligent vehicle controller data files using low bandwidth wireless radio networks by reaching a compression rate of an order of magnitude better than that of the general purpose compressors.

Despite compression time is not critical in the particular application; it would be nice to reduce compression time to the level of the general purpose compressors. The time-intensive part of the compression process is the search for repetitions (enhanced RLE, differential encoding). This takes now $O(nm)$ time where n is the length of the file and m is the maximal seeking distance. [7] presents a new algorithm that does not always compress optimally but uses only $O(n)$ time by applying hash tables and reaches quite good results.

As Table III shows, general purpose compressors reach better compression rates in non-typical use cases and often they can compress the output of VasComp successfully even in typical cases. As these compressors are based on the LZW algorithm, it would be worth developing a special version of LZW that can be decoded by the low-performance controllers as well and build this into the current algorithm.

With these improvements VasComp would be excellent in non-typical use cases as well.

ACKNOWLEDGMENT

The authors are grateful to their colleagues at the Department of Automation and Applied Informatics, Budapest University of Technology and Economics for their advice and encouragement. They also thank Vultron Hungary for providing the test files and the appropriate documents.

REFERENCES

[1] Sándor Juhász, *VAS On-board Database Editor. User Description Summary.* Budapest: Vultron Software Rt., 2003. (internal document)
[2] Gilbert Held, *Data compression. Techniques and applications. Hardware and software considerations.* Chichester [etc.]: Wiley, 1983.
[3] Stephen S. Ruth, Paul J. Kreutzer, "Data Compression for Large Business Files" in *Datamation*, vol. 18, no. 11, pp. 617-623, 1972.
[4] J. A. Storer, T. G. Szymanski, "Data compression via textual substitution" in *Journal of the ACM*, vol. 29, no. 4, pp. 928-951, 1982.
[5] David Solomon, *Data compression: the complete reference*. New York: Springer-Verlag, 1997.
[6] J. Ziv, A. Lempel, "A universal algorithm for sequential data compression" in *IEEE Transactions on Information Theory,* vol. 23, no. 3, pp. 337-343, 1977.
[7] Miklós Ajtai, Randal Burns, Ronald Fagin, Darrell D. E. Long, Larry Stockmeyer, "Compactly Encoding Unstructured Inputs with Differential Compression" in *Journal of the ACM*, vol. 49, no. 3, pp. 318–367, 2002.