POLSKIE TOWARZYSTWO INFORMACJI PRZESTRZENNEJ
ROCZNIKI GEOMATYKI 2014 ○ TOM XII ○ ZESZYT 3(65): 289–301

LOSSLESS COMPRESSION METHOD FOR ASCII UTM FORMAT SEA SURVEY DATA OBTAINED FROM MULTIBEAM ECHOSOUNDER

OPRACOWANIE BEZSTRATNEJ METODY KOMPRESJI DANYCH SONDAŻOWYCH POCHODZĄCYCH Z SONDY WIELOWIĄZKOWEJ ZAPISANYCH W FORMACIE ASCII UTM

Wojciech Maleika, Piotr Czapiewski

Zachodniopomorski Uniwersytet Technologiczny w Szczecinie, Wydział Informatyki

Keywords: multibeam echosounder (MBES), bathymetry, sea survey, UTM coordinate system, data compression, differential coding
Słowa kluczowe: echosonda wielowiązkowa (MBES), batymetria, sondaż morski, system UTM, kompresja danych, kodowanie różnicowe

Introduction

Contemporary hydrographic measurements increasingly often produce immense amounts of measurement data, which are postprocessed using specialised software. The measurement devices used during sea surveys, such as multibeam echosounders, perform readouts of millions of points during one survey (Maleika, Czapiewski, 2013). Due to the huge amount of acquired data we are able to create very accurate seabed models (DTM) (Stephens, Dusing, 2014).

Based on gathered measurement data Digital Terrain Models (DTM) are created, which form the basis for further processing, creating maps or seabed formation visualisation (Borkowski, 2012; Maleika, 2013). Nevertheless, the source data (measurement data) is often saved to mass storage in order to allow for future creation of new models (e.g. with different parameters) (Chybicki et al., 2010; Łubczonek, 2006; Stateczny et. al., 2010; Moszyński et al., 2013).

Developers of hydrographic software usually save such data in ASCII files, which start with a header describing survey parameters, followed by subsequent measurement points in x, y, z format (x, y – point location, e.g. in UTM format, z – depth value measured at the respective point) (Herzfeld et al., 1999). Exemplary fragment of such a file is presented in Figure 1.

```
*** Neptune Ascii file from Kongsberg Simrad A/S ***
Survey name: Dok_5_ZUT_6
Processing operator name: b.u.
Datum: WGS84
Half axis: 6378137.0000000
Flattening: 1/298.25722356300
Coordinate system: utm
Y min.:   5923098.14
X min.:    473086.42
Y max.:   5923492.77
X max.:    473399.28
Latitude cell size: 0.50 meter
Longitude cell size: 0.50 meter
All points, Vert. error

456397.403970  5962579.781657   -7.930000
456397.570143  5962579.947004   -7.680000
456397.975283  5962580.315961   -7.380000
456397.288511  5962581.085974  -11.360000
456397.012521  5962581.752367  -11.470000
456397.154572  5962580.685425  -11.560000
456397.189005  5962580.078541  -11.380000
456397.254514  5962580.745847  -11.250000
```

Figure 1. Fragment of measurement data file created using Neptun application by Kongsberg Simrad

As a result of storing millions of measurement points, such files reach significant sizes. For example, one of test files used in experiments, called gate.utm, consists of 3 812 445 rows, and its total size is 167 747 688 (approximately 160 MB). Each measurement point is stored therein using 44 bytes.

Data file analysis

Closer analysis of source data files brings the conclusion, that a significant information redundancy occurs therein. It can be noted, that:

- data is stored as text;
- each row contains coordinates of a single point;
- each point consists of:
  - a number representing the distance (in metres) north of the base point of UTM zone;
  - the distance (in metres) east of the base point of UTM zone;
  - the depth in meters;
- the numbers describing the location are stored in fixed point format with six fractional digits;
- the depth is stored with two fractional digits, but the number is zero-padded to full six-digits length (additional 4 zeros).

The Universal Transverse Mercator (UTM) projected coordinate system uses a 2-dimensional Cartesian coordinate system to give locations on the surface of the Earth. It is a horizontal position representation, i.e. it is used to identify locations on the Earth independently of vertical position, but differs from the traditional method of latitude and longitude in several respects.

Given that the horizontal accuracy of highest class measurement devices, and consequently the accuracy of stored point locations, can reach several centimetres at best, storing

Table 1. Basic characteristics of test data files

Name	File size [bytes]	No of points	File type
Anchorage	696 312 232	15 825 278	ASCII UTM
Swinging	166 280 796	3 779 109	ASCII UTM
Gate	154 205 876	3 504 679	ASCII UTM
Wrecks	4 874 540	110 785	ASCII UTM

the coordinates with 6 digit precision (i.e. 0.0001mm) is pointless. The information on point locations does not get more accurate, and the data redundancy is significant. Taking the problems characteristics into account (measurements and creation of DTMs) storing the location with the precision of 1 mm (i.e. two fractional digits) should be considered sufficient, and removing digits at further decimal places should not lead to any information loss. Accordingly, when saving depth data it should be sufficient to store up to 2 fractional digits (precision of 1 cm), as this is the accuracy expected from measurement devices. Removing trailing zeroes does not change the accuracy of stored depth information.

Utilising conventional lossless compression algorithms for such data files should bring a high compression ratio (the file contains mostly digits, including many repeating ones), however the specific structure reorganisation and removing redundant data might result in further increase of compression ratio, while still keeping computation time at a reasonable level.

In the following experiments 4 measurement data files were processed (measurements were performed by Maritime Office in Szczecin using Simrad EM3000 echosounder). Table 1 presents basic characteristics of these test files.

In case of ASCII files the information on measurement points locations is not always stored in UTM format, also the format utilizing geographical coordinates is quite often used (in various combinations using degrees, minutes and seconds). The method presented in this paper might be easily adopted for the compression of such a type of data, or of any other data stored in ASCII format complying with the following general pattern: *position_x*, *position_y*, *depth*. The benefits coming from application of the method (the compression ratio) should be similar.

Data reduction methods

Data reduction by discarding redundant information

As described in section "Data file analysis", data files under consideration often contain redundant information. Storing the location with 6 fractional digits or the depth with additional trailing zeroes seems pointless. Those superfluous data is probably a result of utilizing high precision variables in data processing algorithms implementations (including interpolation), which are later on saved to data files with full precision. Figure 2 presents the file contents before and after discarding redundant data.

456397.403970	5962579.781657	−7.930000	456397.403	5962579.781	−7.93
456397.570143	5962579.947004	−7.680000	456397.570	5962579.947	−7.68
456397.975283	5962580.315961	−7.380000	456397.975	5962580.315	−7.38
456397.288511	5962581.085974	−11.360000	456397.288	5962581.085	−11.30
456397.012521	5962581.752367	−11.470000	456397.012	5962581.752	−11.47
456397.154572	5962580.685425	−11.560000	456397.154	5962580.685	−11.56
456397.189005	5962580.078541	−11.380000	456397.189	5962580.078	−11.38
456397.254514	5962580.745847	−11.250000	456397.254	5962580.745	−11.25

Figure 2. Fragment of data file contents before (left) and after (right) discarding redundant data

Table 2. Compression ratio for test data files (discarding redundant data)

Name [bytes]	Source file size	Compressed file size [bytes]	Compression ratio [%]
Anchorage	696 312 232	538 059 452	77,3
Swinging	166 280 796	128 489 706	77,3
Gate	154 205 876	119 159 086	77,3
Wrecks	4 874 540	3 766 690	77,3

As a result of such a simple procedure, without any information loss the compression factor of 77% is obtained (each measurement takes up 34 bytes instead of 44). The results of such a compression of test data files are presented in Table 2.

Various hydrographic software stores data in ASCII files in a different way (with varying precision / fractional digits number). One should aim at such an information storing scheme, that results in storing location and depth with 1 mm and 1 cm precision respectively.

Data conversion to binary format

Plain text format is readable for the user, however, from computer's perspective operating on data stored as text is highly ineffective – each time the data is loaded, the software needs to convert text into a number. By storing binary data we utilize computer memory more effectively and we gain faster access to particular measurement points, which translates into faster file processing operations. The data file itself, due to a much more effective saving format, is significantly smaller in size.

When storing binary data, the following floating point types should be used: double precision number to describe the location (two 64-bit numbers) and single precision number to describe the depth (one 32-bit number). In such case a single measurement point takes up 20 bytes. Figure 3 presents the data file contents before and after saving in binary format.

456397.403	5962579.781	−7.93
456397.570	5962579.947	−7.68
456397.975	5962580.315	−7.38
456397.288	5962581.085	−11.30
456397.012	5962581.752	−11.47
456397.154	5962580.685	−11.56
456397.189	5962580.078	−11.38
456397.254	5962580.745	−11.25

31	08	AC	9C	35	DB	1B	41	6D	E7	FB	F1	D4	BE	56	41
8F	C2	FD	C0	7B	14	AE	47	36	DB	1B	41	E3	A5	9B	FC
D4	BE	56	41	8F	C2	F5	C0	66	66	66	E6	37	DB	1B	41
C3	F5	28	14	D5	BE	56	41	F6	28	EC	C0	D5	78	E9	26
E1	DA	1B	41	D7	A3	70	45	D0	BE	56	41	8F	C2	35	C1
5E	BA	49	0C	E4	DA	1B	41	9C	C4	20	70	D0	BE	56	41
1F	85	37	C1												

Figure 3. The contents of an ASCII file (left) and a binary file (right)

As it turns out, storing the data in question in a binary format allows to obtain a compression ratio of 45%. The results of such a compression of test data files are presented in Table 3.

Storing measurement data in binary files is highly advisable (many software packages offer such a functionality), and using ASCII files can be justified for import/export between heterogeneous software.

Table 3. Compression ratio for test data files (binary format)

Name	Source file size [bytes]	Compressed file size [bytes]	Compression ratio [%]
Anchorage	696 312 232	316 505 560	45.5
Swinging	166 280 796	75 582 180	45.5
Gate	154 205 876	70 093 580	45.5
Wrecks	4 874 540	2 215 700	45.5

It should be noted, that many hydrographic software packages offer the possibility to store data in several binary formats, such as ALL, GMT GRD / NetCDF, Etopo2 and 5, USGS DEM, CARIS HDCS, SHOALS, HTF and others. The vendors not always provide a detailed description of a given data format, however, it can be safely assumed, that those formats consists of binary entries with the location and depth of subsequent measurement points (possibly along with some additional information). The benefits stemming from utilizing such binary storage formats should be similar to those described above (the compression factor of approximately 2). The authors have not come across any reference (in scientific literature or in official brochures from hardware/software vendors) to using the differential coding in any format describing MBES data, which is a major contribution of this paper. It is most probable, that many of the existing binary formats might be modified using the above described algorithms, leading to achieving similar benefits (compression ratio). It would however require to develop a separate algorithm for each of those formats.

Furthermore, certain data formats exist, that store significantly broader scope of measurement information. For example, the s7k format (a record-based data format defined for data logging and network transmission for use, in part, with the SeaBat™ 7k systems) stores the data in records of various types. Those records may contain the information on sonar setup, geometry, seabed detection, image from a side scanner, bathymetry data and others. The development of a compression method for such data would require a different approach for each of the data formats.

Data compression using LZW algorithm

Given the characteristics of ASCII UTM data files and of binary data files storing the same kind of information, it should be expected, that a significant compression ratio can be achieved using known lossless compression methods, e.g. LZW algorithm (Ziv, Lempel, 1977). In order to verify this hypothesis, the test files were compressed using the most popular ZIP software, which performs lossless compression based on LZW algorithm (Grabmayer et al., 2012). The results of the experiment, including compression ratio and processing time, are presented in Table 4.

Over five times decrease in size could be considered satisfactory in many cases. The LZW algorithm efficiently detects and compresses repeating digits and spaces in case of ASCII files or repeating number values in case of binary files (unused bytes in floating point format). It should be noted, that ASCII files can be compressed better than binary files, but at the expense of approximately twice the computation time.

Table 4. Compression ratio and processing time for test files compressed using ZIP software

Name	Source file size [bytes]	Compressed ASCII file size [bytes]	Compression ratio [%] and time [s]	Compressed binary file size [bytes]	Compression ratio [%] and time [s]
Anchorage	696 312 232	89 824 270	12.9 (1020)	127 425 134	18.3 (460)
Swinging	166 280 796	29 597 994	17.8 (250)	32 923 598	19.8 (113)
Gate	154 205 876	24 827 144	16.1 (237)	29 453 329	19.1 (93)
Wrecks	4 874 540	589 821	12.1 (8)	901 792	18.5 (3)

Differential data coding using varying byte length

The next step of research consisted in examining the possibility of achieving even higher compression factor by introducing a specific reorganisation of data file, including differential coding of subsequent location and depth values. Differential coding allows for a more effective utilisation of memory and a significant diminishing of file size, hence the differences between subsequent measurement points may be stored using smaller number of bytes. Here we propose the following approach:

- In the first stage fixed point numbers are converted to integer numbers by removing the decimal point (or more formally: by multiplying location values by 100 and depth values by 10), which is presented in Figure 4.

456397.403	5962579.781	−7.93	456397403	5962579781	−793
456397.570	5962579.947	−7.68	456397570	5962579947	−768
456397.975	5962580.315	−7.38	456397975	5962580315	−738
456397.288	5962581.085	−11.30	456397288	5962581085	−1130
456397.012	5962581.752	−11.47	456397012	5962581752	−1147
456397.154	5962580.685	−11.56	456397154	5962580685	−1156
456397.189	5962580.078	−11.38	456397189	5962580078	−1138
456397.254	5962580.745	−11.25	456397254	5962580745	−1125

Figure 4. Data file contents before (left) and after the conversion of real numbers to integers (right)

- In the second stage all the rows after the first one are converted as follows: the difference between the current and previous row values is calculated and stored instead of the actual value (separately for x, y, z). The result of this step is illustrated in Figure 5.

456397.403	5962579.781	−7.93	456397403	5962579781	−793
456397.570	5962579.947	−7.68	167	166	25
456397.975	5962580.315	−7.38	405	368	30
456376.288	5962561.085	−11.36	-687	770	-392
456377.012	5962561.752	−11.47	-276	667	-17
456397.154	5962580.685	−11.56	142	-1067	-9
456377.189	5962562.078	−11.38	35	-607	18
456377.254	5962562.745	−11.25	65	667	13

Figure 5. Data file contents before (left) and after the differential coding (right)

- In the third stage the values of subsequent measurements are additionally encoded. Since most of the values are small, it could be beneficial to use a technique of storing the numbers using variables of varying length. Obviously, the information about the actually used number of bytes must be appended. For this purpose Variable Length Value Coding method (Cormack, Horspool, 1984] was utilised. Each VLV value is stored in bytes, whereas each byte contains two portions: 7 bits contain the actual information and 1 bit denotes continuation. If the most significant bit (continuation bit) is set, then the number is continued in the next byte. Otherwise, this is the last byte of a number. In order to encode a number in VLV, it needs to be divided into 7-bit long groups; then each group is appended with the continuation bit. In such case all the numbers within the range <-63; 64> are stored using one byte, numbers within the range <-16383; 16384> using two bytes, and so on. In order to retrieve a number encoded using VLV, the continuation bit must be removed, remaining bits must be concatenated to the number being formed, until the final byte is encountered.

Storing the numbers using Variable Length Value Coding leads to a significant decrease in data file sizes. Table 5 presents a comparison of raw data structure size and after VLV encoding.

Table 5. Data structure size when using VLV technique

Data	ASCII UTM [bytes]	Binary file [bytes]	Variable Length Value Coding [bytes]
Coordinate x	~ 28	8	1-3 (frequent), 4-8 (rare), >8 (very rare)
Coordinate y	~ 28	8	1-3 (frequent), 4-8 (rare), >8 (very rare)
Depth z	~ 10-11	4	1-2 (frequent), 3-4 (rare), >4 (very rare)

Below the algorithm is presented developed for the purpose of ASCII UTM files compression by differential encoding and VLV coding.

```
IN:Source file in, OUT destination file out
procedure EncodeFile(in, out)
. Read floating point numbers
    in.read(X);
    in.read(Y);
    in.read(Z);
. Stage I: Convert to integer numbers and truncate
    X ← b X · 1000 c;
    Y ← b Y · 1000 c;
    Z ← b Z · 100 c;
    First X ← X;
    First Y ← Y;
. Create temporary values
    First Z ← Z;
. Stage II: calculate and store the differences
    while in.eof() = false do          . Until the end of file is reached
        in.read(Nx);
        in.read(Ny);
        in.read(Nz);
        Nx ← b Nx · 1000 c;
        Ny ← b Ny · 1000 c;
        Nz ← b Nz · 100 c;
. Add the differences diff x, diff y, diff z to the array
        differences.add(Nx - X, Ny - Y, Nz - Z);
        X ← Nx;
        Y ← Ny;
        Z ← Nz;
    end while
. How many bytes are required for the first difference, for each variable
    diff size x ← GetDiffSize(differences[0].diff x);
    diff size y ← GetDiffSize(differences[0].diff y);
    diff size z ← GetDiffSize(differences[0].diff z);
    num x ← 0;                         . Counters for the respective differences
    num y ← 0;
    num z ← 0;
. Stage III: count the differences with respect to the number of bytes and store them
    for i ← 1 to differences.size -1 do
        num x ← num x + 1;
        num y ← num y + 1;
        num z ← num z + 1;
. Does the difference fit into the given number of bytes?
```

```
. Can it be stored using less bytes?
        if CheckDiffSize(diff size x, differences[i].diff x) = true then
. Store information on differences sizes and count into the array
            encode x.add(diff size x, num x);
            num x ← 0;
            diff size x ← GetDiffSize(differences[i].diff x);
        end if
        if CheckDiffSize(diff size y, differences[i].diff y) = true then
            encode y.add(diff size y, num y);
            num y ← 0;
            diff size y ← GetDiffSize(differences[i].diff y);
        end if
        if CheckDiffSize(diff size z, differences[i].diff z) = true then
            encode z.add(diff size z, num z);
            num z ← 0;
            diff size z ← GetDiffSize(differences[i].diff z);
        end if
    end for
    encode x.add(diff size x, num x + 1);
    encode y.add(diff size y, num y + 1);
    encode z.add(diff size z, num z + 1);
. Save the first values to a file in binary format
    out.write(differences.size, 4);
    out.write(First X, 8);
    out.write(First Y, 8);
    out.write(First Z, 8);
    diff x ← 0;
    diff y ← 0;
    diff z ← 0;
    num x ← 0;
    num y ← 0;
    num z ← 0;
. Iterators for respective arrays
    for i ← 0 to differences.size -1 do
        if num x = 0 then          . Get the count of differences and size
            diff size x ← encode x[diff x].bytes required;
            num x ← encode x[diff x].num of elements;
            diff x ← diff x + 1;
            SaveVLV(out, diff size x, num x);
        end if
        if num y = 0 then
            diff size y ← encode y[diff y].bytes required;
            num y ← encode y[diff y].num of elements;
            diff y ← diff y + 1;
            SaveVLV(out, diff size y, num y);
        end if
        if num z = 0 then
            diff size z ← encode z[diff z].bytes required;
            num z ← encode z[diff z].num of elements;
            diff z ← diff z + 1;
            SaveVLV(out, diff size z, num z);
        end if
. Save the differences to a file, using a given number of bytes, in binary format
        out.write(differences[i].diff x, diff size x);
        out.write(differences[i].diff y, diff size y);
        out.write(differences[i].diff z, diff size z);
    end for
end procedure
```

The decoding process is very similar. In the first stage subsequent bytes are read from the encoded file and then decoded. After initial decoding the numbers are converted from differential coding into plain values (actual measurement values).

```
IN:Source file in, OUT destination file out
procedure DecodeFile(in, out)
. Read 4-byte integer number (binary)
    in.read(LinesCount, 4);
. Read first values stored using 8 bytes
    in.read(X, 8);
    in.read(Y, 8);
    in.read(Z, 8);
. Output the read values in text format
    out.write(X div 1000, ".", X mod 1000, " ");
    out.write(Y div 1000, ".", Y mod 1000, " ");
    out.write(Z div 100, ".", abs(Z) mod 100, " \n");
. Read the number and size of differences
    LoadVLV(in, num x, diff size x);
    LoadVLV(in, num y, diff size y);
    LoadVLV(in, num z, diff size z);
    for i ← 1 to LinesCount do
        if num x = 0 then
            LoadVLV(in, num x, diff size x);
        end if
        if num y = 0 then
            LoadVLV(in, num y, diff size y);
        end if
        if num z = 0 then
            LoadVLV(in, num z, diff size z);
        end if
. Read the differences stored in binary format as integer numbers, using a given number of bytes
        in.read(dx, diff size x);
        in.read(dy, diff size y);
        in.read(dz, diff size z);
        X ← X + dx;
        Y ← Y + dy;
        Z ← Z + dz;
. Calculate new values and output in text format
        out.write(X div 1000, ".", X mod 1000, " ");
        out.write(Y div 1000, ".", Y mod 1000, " ");
        out.write(Z div 100, ".", abs(Z) mod 100, " \n");
        num x ← num x -1;
        num y ← num y -1;
        num z ← num z -1;
    end for
end procedure
```

As a result of a cycle of coding/decoding operations we obtain the same file as the input, hence we deal with the lossless data compression.

The effectiveness evaluation of the proposed algorithm is presented in Table 6.

Table 6. Compression ratio and computation time for the files saved using differential and VLV encoding

Name	Source file size [bytes]	Compressed file size [bytes]	Compression ratio [%]	Compression time [s]
Anchorage	696 312 232	70 327 498	10.1	128.6
Swinging	166 280 796	19 787 450	11.9	30.7
Gate	154 205 876	15 266 374	9.9	29.9
Wrecks	4 874 540	419 210	8.6	1.0

Comparison of the above results to the ones obtained using LZW method shows, that the compression ratio when using differential and VLV coding is slightly better and computation time shorter. This leads to a conclusion, that the method based on differential coding combined with coding using varying number of bytes is well adjusted to the characteristics of the sea survey measurement data. The theoretical low boundary for the compression factor is approximately 6.8%, which could be obtained when all the subsequent (x, y, z) numbers in a file are encoded using single bytes (3 bytes for a measurement point).

LZW compression of differential data stored using varying number of bytes

The data obtained after conversion to differential form and stored using VLV algorithm could be further processed by LZW compression, in order to minimize the redundancy in data. In order to verify the validity of such an approach, the compression of test files encoded as described in previous section „Data compression using LZW algorithm" was carried out. The results are presented in Table 7.

Due to additional application of LZW algorithm, further reduction of data size was achieved (50-70% smaller size than in the previous step). After applying all the proposed techniques, i.e. differential coding, encoding using varying number of bytes and LZW compression, the final compression factor of 5-10% can be obtained for measurement data files. Hence, the reduction of data size by a factor of 10-20 was achieved. It should be noted, that processing time is shorter than for pure LZW compression. This could be explained by the fact, that the amount of data sent as the input to LZW algorithm is significantly smaller due to initial reduction by the differential and VLV encoding, and those two algorithms are much faster than LZW.

Table 7. Compression ratio and computation time for the data files stored using differential coding, VLV coding and LZW compression

Name	Source file size [bytes]	Compressed file size [bytes]	Compression ratio [%]	Compression time [s]
Anchorage	696 312 232	44 563 986	6.4	210.3
Swinging	166 280 796	14 466 430	8.7	52.0
Gate	154 205 876	11 102 824	7.2	47.8
Wrecks	4 874 540	258 347	5.3	1.7

Conclusions

In Figure 6 the summary of the research results is presented in terms of the compression factor obtained for test data files using different encoding methods.

Measurement data files obtained as a result of sea surveys performed using a multibeam echosounder are quite considerable in size, since they contain millions of measurement points. Saving this data is not problematic, however it should be noted, that storing or transmitting such huge data sets can be cumbersome. In certain cases introducing a lossless compression method is definitely advisable. Utilising popular universal compression algorithms (such as LZW implemented in the ZIP compressor) gives good results, reducing the data volume by a factor of up to five.

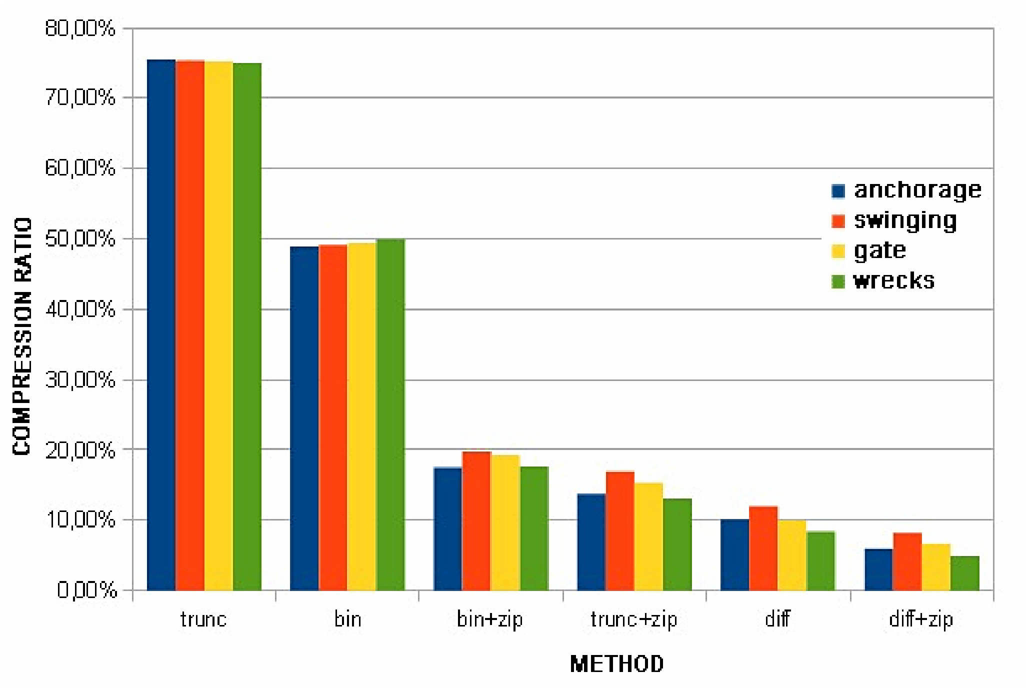

Figure 6. Effectiveness comparison of presented compression methods for ASCII UTM data

The method proposed in this paper consists in initial reorganisation of data file by discarding redundant information, followed by calculating differences between measurement points and encoding them using varying number of bytes, and finally compressing using LZW algorithm. Such a procedure leads to significantly improved compression results, both in terms of compression ratio and processing time. The compression ratio reaches 5-10%, which means reduction by order of magnitude. The whole procedure is reasonably fast, overall processing time is shorter than for ZIP compression alone.

The developed algorithm may be used in hydrographic software as additional functionality for saving source measurement data. Its utilisation may significantly reduce the amount of stored data, faster data transfer in computer networks, while still maintaining acceptable compression time.

It should be clearly emphasized, that the presented method of differential binary data coding along with the additional ZIP compression could be easily adapted for the compression of other existing file formats. In case of the files containing similar information (location, depth) the obtained benefits (compression ratio) should be probably comparable. The authors focused on one of the commonly used formats in order to perform a broad spectrum of tests and to develop a detailed compression algorithm. This also allowed to perform a reliable comparison of the results obtained using different compression methods.

It seems purposeful for hardware and software vendors to look for technical solutions allowing for a significant reduction of data (measurement data and DTM data), while preserving the detailed bathymetric information.

References

Borkowski P., 2012: Data fusion in a navigational decision support system on a sea-going vessel. Polish Maritime Research vol. 19, no. 4(76): 78-85.

Chybicki A., Łubniewski Z., Moszyński M., 2010: Using wavelet techniques for multibeam sonar bathymetry data compression. *Hydroacoustics* vol. 13, no. 3: 31-38.

Cormack G.V, Horspool R.N., 1984: Algorithms for adaptive Huffman codes. *Information Processing Letters* vol.18, no. 3: 159-165.

Grabmayer C., Endrullis J., Hendriks D., Klop J.W., Moss L.S., 2012: Automatic Sequences and Zip-Specifications. 27th Annual ACM/IEEE Symposium On Logic In Computer Science (LICS), Book Series: IEEE Symposium on Logic in Computer Science: 335-344, DOI: 10.1109/LICS.2012.44.

Herzfeld U.C., Matassa M.S., Mimler M., 1999: A Program for Matching Universal Transverse Mercator (UTM) and Geographic Coordinates. *Computers & Geosciences* vol. 25, no. 7: 765-773, DOI: 10.1016/S0098-3004(99)00020-5.

Łubczonek J., 2006: Analiza porównawcza metod modelowania powierzchni w aspekcie opracowania numerycznego modelu dna morskiego. *Roczniki Geomatyki* t. 4, z. 3: 151-163, PTIP Warszawa.

Maleika W., Czapiewski P., 2013: Visualisation of multibeam echosounder measurement data. [W:] Maji P. et al. (Eds.) Pattern Recognition and Machine Intelligence, *Lecture Notes in Computer Science* vol. 8251: 373-380, Springer-Verlag, Berlin Heidelberg.

Maleika W., 2013: The influence of track configuration and multibeam echosounder parameters on the accuracy of seabed DTMs obtained in shallow water. *Earth Science Informatics* 6: 47–69, DOI 10.1007/s12145-013-0111-9.

Moszyński M., Chybicki A., Kulawiak M., Łubniewski Z., 2013: A novel method for archiving multibeam sonar data with emphasis on efficient record size reduction and storage. *Polish Maritime Research* no. 1(77), vol. 20.

Stateczny A., Grodzicki P., Włodarczyk M., 2010: Badanie wpływu parametrów filtracji geodanych pozyskiwanych wielowiązkową sondą interferometryczną GeoSwath+ na wynik modelowania powierzchni dna. *Roczniki Geomatyki* t. 8, z. 5: 121-130, PTIP Warszawa.

Stephens D., Diesing M., 2014: A comparison of supervised classification methods for the prediction of substrate type using multibeam acoustic and legacy grain-size data. *PloS ONE* vol. 9, no. 4.

Ziv J., Lempel A., 1977: Universal Algorithm for Sequential Data Compression, IEEE *Transactions on Information Theory* vol.23, no. 3: 337-343.

Abstract

Data gathered through seabed surveys performed using multibeam echosounder tend to be significant in size. Quite often a single measurement session leads to obtaining even several million distinct points (usually in x, y, z format). These data are saved in files (often text files), where x, y represent the location of a point (in geographical format, or more commonly in UTM format) and z represents the measured depth at the respective point. Due to the huge amount of such points, the data occupy a significant space in memory or in storage system (the order of megabytes for small areas and of gigabytes for larger ones). The paper contains a survey of existing methods of compressing ASCII UTM files and a proposal of a novel method tailored for a particular data structure. As a result of utilising differential coding and coding using varying length values, the size of such files can be diminished by a factor exceeding ten, while preserving the full information. The paper presents a detailed description of the proposed algorithm and experimental results using real data.

Streszczenie

Dane pozyskane z sondaży dna morskiego wykonane z użyciem sondy wielowiązkowej cechują się znacznym rozmiarem. Bardzo często w wyniku jednej sesji pomiarowej otrzymujemy nawet kilka milionów pojedynczych punktów (najczęściej w formacie x,y,z*). Informacje te zapisywane są w plikach, często tekstowych, gdzie* x,y *to położenie punktu (w formacie geograficznym lub częściej UTM),* *a* z *określa zmierzoną głębokość w tym punkcie. Ze względu na ogromną liczbę tych punktów dane te zajmują w pamięci komputera lub na dyskach znaczny rozmiar (liczony w MB dla małych obszarów lub GB dla większych). Autorzy przedstawili w artykule różne metody kompresji plików ASCII UTM, w tym opracowaną autorską metodę dopasowaną do struktury danych. Dzięki zastosowaniu metody zapisu różnicowego z wykorzystaniem zmiennej długości w bajtach możemy ponad dziesięciokrotnie zmniejszyć rozmiary tego typu plików, przy zachowaniu pełnej informacji. W artykule przedstawiono szczegółowy algorytm oraz testy wykonane na danych rzeczywistych.*

dr inż. Wojciech Maleika
wmaleika@wi.zut.edu.pl

dr inż. Piotr Czapiewski
pczapiewski@wi.zut.edu.pl