Data compression for modern coding for Z80

Development

Many programs made on the Spectrum in the 1980s did not use data packaging technology. In general, compression was not so widely known, there were few packers. RLE was not considered a shameful compression algorithm! Now the situation has changed; almost every modern release, at least, will come in packaged form, and as a maximum - will keep some of the data packed and unpack it on the fly, as needed. However, talking about compressors often shows that a rare encoder sees the big picture. People have been using the same packer for decades; sometimes a good, just outdated packer, sometimes a packer that you shouldn't even start using 10 years ago. I made a series of tests for ten fashionable packers enough to try to figure out which packer to take for a modern application. I hope

Introduction

In general, I want to start with the fact that the most active use of packers in the 1990s - the repackaging of cassette and disk releases - does not interest me very much. I look at packers in the sight of using them to pack data in my programs (most often - demah). When I talk about unpacking data, I most likely think about unpacking on the fly, right while the program is running, and not once, right after downloading. A lot of packers bear the imprints of old habits: unpackers attach to the data, try to provide for the relocation of decompressors and code blocks at the end of the download, etc. In this article, I will ignore features of this kind. I'm only interested in packing the data, if possible, without unnecessary headers and identifiers, and unpacking is not more complicated than something like:

```
ld hl,PackedData : ld de,DestinationAddress : call Decompress
```

I mean that our unpacker does not require a significant amount of additional memory for its work (Exomizer uses a buffer of 156 bytes, the rest handle it this way). I also mean that the unpacker should do nothing but stupidly unpacking. In addition, since we are talking about *modern* coding, I mean that the data is not packaged on the Spectrum. Each packer included in the review can pack files on a PC (as a rule, maximum packing is achieved only on a PC). In addition, since we code on Z80, each packer will have at least one unpacker for Z80.

For testing, I prepared a new datasetto better understand the strengths and weaknesses of each packer. Because I'm interested in packaging for the Z80, I deliberately excluded files larger than 64Kb. All cooked files are divided into 5 groups. The first two groups - “calgary” (8 files) and "canterbury" (5 files) include all files smaller than 64K from standard compression cases (full cases can be found here) The graphics group contains 30 images for the Spectrum, mostly modern, of varying degrees of complexity (compressibility), and of various types, including various multigigacolors and borders. The music group contains 24 music files for various music editors, including several beeper tracks. Finally, the "mixedzx" group contains 10 files with mixed data (7 unpacked games, one unpacked demo and two partially packed demos). I wanted to provide packers with a variety of typical "live" data, with varying degrees of compressibility, and see how packers cope with this task.

There is one more point that, it seems to me, not everyone understands. Data packaging is a special kind of transcoding, and for most packers, this transcoding can be done in different ways. Roughly speaking, the same packer can pack the same data for better or worse. For many compression algorithms, it is easier to implement the optimal packer, i.e. a packer that will always produce the minimum size of packed data for its format. Optimality is rarely achievable in modern archivers, due to the high complexity of the algorithms used and the very low optimal compression rate, especially for large files. Optimality is almost always achievable in packers for 8-bit machines, as we have relatively small files and relatively primitive compression algorithms. The difference between optimal and non-optimal compression can be quite significant. For example, the optimal packer for the MegaLZ format was written much earlier than the optimal packer for the Hrust 1 format, so on the forums you can find the results of old tests in which MegaLZ compresses on average better. In this test, I tried to clarify what kind of packer is used, and I will definitely mention it in the text to understand what differences are related to the compression format and which are related to the shortcomings of existing packers.

Information on Compressors Included in the Overview

I will say right away that I do not pretend that I know each of these packers in detail. I have been working with some of them for several years, some of which I poked only now to prepare a review. Information is not always complete; I could be wrong in some ways. Therefore, I will be glad if someone can supplement the information available to me or correct any inaccuracies in the descriptions.

ApLib is a universal data packaging library for weak machines written by Jørgen Ibsen, here is its official page. The library is old, was written in the second half of the 1990s. According to the algorithms - a variation of LZSS with a large number of additional tricks (for example, it contains codes for reusing recently encountered offsets, as in LZX. Such a dependence of the codes on the context greatly complicates the optimal packaging). The standard appack.exe packer, which comes with the library, adds a header to the packed data, so no one uses it. Instead, they use one of the packer assemblies that does not write headers, for example appack_raw.exe from the SpritesMind Seg forumcollected by our compatriot r57shell. The standard packer is not optimal (it presses slightly worse than the optimal Hrust 1 packer), but fortunately, r57shell wrote a new packer closer to the optimal (although still not optimal) packer, which I renamed appack_r57shell.exe for convenience. It is already significantly ahead of Hrust 1 in compression. The latest packer version is here . Packaging is done like this:

```
appack_raw.exe c inputfile [outputfile.aplib]
```

or

```
appack_r57shell.exe inputfile [outputfile.aplib]
```

Since packers are not optimal, in theory, either of the two can win on separate files; in my tests I always used only the r57shell packer (since it almost always wins).

The first, very braking unpackers on the Z80 were written by Dan Weiss for the TI83 calculator and Maxim for the Sega Master System; Really good unpackers have already been written for the Spectrum Metalbrain, Utopian and Antonio Villena, see the thread where they were published . I included 4 unpackers in this test.

1. aplib156b.asm is a compact unpacker, totaling 156 bytes, using the registers AF, BC, DE, HL, IX and, attention, IY. On average, it decompresses one byte for 165 cycles of the Z80.
2. aplib197b.asm - the unpacker is faster, takes 197 + 2 bytes and uses the registers AF, AF ', BC, DE, HL, IX and, attention, IY. It decompresses one byte in 106 cycles of the Z80. Attention! This and the next two unpackers that use AF 'contain an unpleasant bug, which is that one of the SBC HL, BC commands implies a C flag that was previously cleared, which requires adding a couple of commands before calling the unpacker:

   ```
   or a : ex af,af'
   ```

3. aplib227b.asm takes 227 + 2 bytes, uses the registers AF, AF ', BC, DE, HL, IX and, attention, IY, and decompresses one byte for 102 cycles of Z80.
4. aplib247b.asm takes up 247 + 2 bytes, uses the registers AF, AF ', BC, DE, HL, IX and, attention, IY, and decompresses one byte per 100 clock cycles.

Exomizer is a modern packer written by Magnus Lind. There is no documentation on the packaging format, but the packer is available in source code . I did not understand the packer’s device and data format and I can’t say whether the packer is optimal (if someone knows, it will be useful to find out). The file can be packed with the command:

```
exomizer.exe raw inputfile -o [outputfile.exo]
```

In addition to the standard format, there is a “simple” data format, which allows the unpacker to be slightly simplified. Data packaging for the "simple" format is carried out as follows:

```
exomizer.exe raw -c inputfile -o [outputfile.exo_simple]
```

The "Simple" format packs about 0.2% worse. The difference in the decompression speed between the two formats is negligible, but the decompressors for the “simple” format are shorter by 15 bytes. I decided not to include the “simple” unpacker test results in this review. In addition, Exomizer can pack data backwards (this is useful for unpacking data to the very top of the memory, where it is impossible to uncover lap with a small margin; the corresponding unpackers deexo_b.asm and deexo_simple_b.asm can be found in the official release.

Exomizer unpackers were written in the same team that optimized ApLib unpackers .

1. deexo.asm is the main, slightly leisurely unpacker. It occupies 169 bytes, plus it requires a memory buffer of 156 bytes for its operation. The unpacker uses AF, BC, DE, HL, IX and, attention, IY. The decompression speed is 287 clocks per decompressed byte.
2. The standard unpacker contains instructions on how to significantly accelerate unpacking by adding only 5 bytes to the unpacker; deexo_plus.asm - implements these recommendations. Unpacker takes 174 bytes; decompression speed - 248 clocks per byte.

Hrum is the first of three Dmitry Piankov packers to get into this review. Hrum was positioned as a packer with good compression and fast decompression. According to the device, it is an LZSS packer of medium complexity. The old Hrum packers for the Spectrum were not optimal, but mhmt.exe from lvd includes the optimal compressor in the Hrum data format, which is called as follows:

```
mhmt.exe -hrm inputfile [outputfile.hrum]
```

The standard version of Hrum on the Spectrum applied the unpacker to the clamped block, and the unpacker was designed to automatically decompose the data in memory and run the code (i.e., the packer was designed to unpack the data downloaded from the tape or disk at a time). For this review, I disassembled the standard Hrum unpacker and adapted it to work with compressed data blocks:

1. unhrum_std.asm - takes 104 bytes, can also be compiled as a 105 byte relocatable unpacker. The unpacker uses AF, BC, DE, HL, BC ', DE' and, attention, HL '; the relocated version uses IX. The decompression speed is 97 cycles per byte.

Hrust 1 - the most effective packer of Dmitry Pyankov. In terms of internal design, Hrust 1 is a sophisticated LZSS packer with a bunch of additional chips, such as copying blocks with holes and dynamically changing link codes (as in LZB). Like Hrum, Hrust 1 was positioned as a packer for simultaneous decompression of downloaded data, with built-in unpacker, relocation, etc. features. However, when lvd implemented the Hrust 1 wrapper in its mhmt, it removed the headers and other supporting data from the packed data, and also prepared the appropriate unpackers. Thus, Hrust 1 can now be used as a general-purpose data compression library. The hrust 1 wrapper in mhmt is not optimal; fortunately, Evgeny Larchenkorecently wrote a new optimal packer for Hrust 1 format called oh1c.exe . This packer adds standard Hrust 1 headers to the compressed data; I took the liberty of slightly finalizing the packer so that it could also issue bare packed data. This packer is called as follows:

```
oh1c.exe -r inputfile [outputfile.hrust1]
```

Data packaged with the -r option can be unpacked using standard mhmt unpackers.

There were 2 Hrust 1 unpackers attached to mhmt:

1. dehrust_ix.asm - a slower unpacker using index registers. It occupies 234 bytes, uses AF, BC, DE, HL, BC ', DE', HL 'and IX, on average it decompresses one byte per 132 processor cycles.
2. dehrust_stk.asm - a faster unpacker, relocatable (!), but, unfortunately, reading data in a stack. It occupies 209 bytes, uses AF, BC, DE, HL, BC ', DE', IX, and, attention, HL ', decompresses the byte in 120 cycles.

Hrust 2 - Dmitry Pyankov's packer, who was positioned to pack texts. Perhaps the old Spectrum packers gave some differences when packing in these two formats. As the tests below show, when using optimal packers, the advantage of Hrust 2 over Hrust 1 in texts is very small, and for mixed data, Hrust 2 loses to Hrust 1 about 0.6%. The packing algorithm is a tricked out LZSS. The optimal packer for Hrust 2 was recently published by Evgeny Larchenko . It is called oh2c.exe, and is called as follows:

```
oh2c.exe inputfile [outputfile.hrust2]
```

This packer creates files with internal data headers of Hrust 2; adding an option to generate bare data will, of course, be easy, but it seemed to me that this was not necessary.

The Hrust 2 format has an official unpacker:

1. DEHRUST_2x.asm takes up 212 bytes, uses AF, BC, DE, HL, BC ', DE' and, attention, HL ', and decompresses the data byte in 127 cycles.

LZ4 is one of the trendy new wave data packers developed by Yann Collet, see project page. This is a byte LZSS packer, yeah, just like the old Titusa packer. Byte packers were sometimes found in the 1990s, but quickly lost to mixed bit-byte packers, as could not compete in compression ratio. Why did they return to them, especially on modern computers? The fact is that the difference in speed between calculations on modern processors and memory access is so great that unpacking compressed data is faster than blunt copying of previously prepared data, i.e. Packing for a very fast unpacker is a way to speed up programs that work with large chunks of data. In general, there are a lot of similar purpose libraries (Snappy, FastLZ, QuickLZ, LZTurbo, LZO, LibLZF). I chose LZ4 due to the fact that, by the standards of such libraries, it packs relatively well and also because that several unpackers for the Z80 have already been written for him. Packing in LZ4 format is best done usingoptimal compressor smallz4.exe , with this call format:

```
smallz4.exe -9 inputfile [outputfile]
```

I tested the following unpackers:

1. unlz4_drapich.asm is the most sophisticated unpacker written by Piotr Drapich (I downloaded it here). It carefully checks the signal bytes and headers of the ZX4 and, apparently, should handle the files in older versions of the format. The payback for such conscientiousness is the relatively large size of the unpacker - 251 bytes. The registers AF, BC, DE, HL, AF 'and IX are used. To justify its reputation on large machines, a very high decompression speed is 33.8 cycles per unpacked byte.
2. unlz4_stephenw32768.asm is a very simplified unpacker of authorship stephenw32768 with WoS , which can work only with packed data cleared from headers. The author of the unpacker wrote a special utility for removing headers, so you have to pack the data in two steps:

   ```
   smallz4.exe -9 inputfile outputfile.lz4
   lz4-extract.exe < outputfile.lz4 > outputfile.lz4raw
   ```

   The resulting binary can already be unpacked. The decompression speed is slightly lower - 34.4 cycles per byte, but the decompressor takes only 72 bytes.
3. At the end of the review, my new unpacker for LZ4 is attached - unlz4_spke.asm, which takes 104 bytes and decompresses data a little faster than unlz4_drapich.asm - 33 clock cycles per unpacked byte, i.e. almost as much as 1.5 * LDIR. My unpacker checks the headers only very superficially, to reduce the size and speed up the work. If desired, the DATA_HAS_HEADERS option can be commented out, which puts the unpacker in the mode of working with bare data without headers, such as the lz4-extract.exe utility prepares. This will slightly reduce the size of the data and the decompressor, as well as slightly accelerate the decompression. The optional ALLOW_USING_IX option will add some speed if you can afford to lose the value in IX.

MegaLZ is a fashion packer developed by fyrex. Technically, it is a medium complexity LZSS packer. As I understand it, thanks to the work of lvd, MegaLZ turned out to be one of the first (first ?!) packer for the Spectrum with the optimal parser (see the mirror of the old home page). This allowed MegaLZ to compete in compression with the best LZ packers on the platform (such as Hrust 1, the optimal packer of which was written only recently). Now there are two optimal packers for MegaLZ: megalz.exe and mhmt.exe. I can’t give a link to the mhmt distribution, because I could not find a single page with the release; The old Google Code page is blank. The packer is called like this:

```
megalz.exe inputfile [outputfile.megalz]
```

or

```
mhmt.exe -mlz inputfile [outputfile.megalz]
```

(the size of the packed file will be the same, regardless of the selected packer).

I know only one unpacker for MegaLZ.

1. DEC40.asm - 110 bytes of relocatable code. The decompression speed is about 131 cycles per decompressed byte. The registers AF, BC, DE, HL and AF 'are used.

In addition, I heard that lvd made modifications to MegaLZ to work with the ring buffer. I did not find anything about this on the net.

Pletter 0.5 - according to several comments on the forums, Pletter is a fairly popular MSX packer. However, there is little information about Pletter on the network. Developed by Pletter by the XL2S Entertainment team in 2007-2008 (see project home page) In fact, this is, in much the same way as the ZX7, the development of the BitBuster compressor, i.e. Another simple bit-byte LZSS packer. However, unlike BitBuster / ZX7, Pletter is trying to be more flexible in its format. In short, the BitBuster storage format is pretty dumb. Nevertheless, it is clearly based on some statistical tests, so if you start to slightly vary the packaging format, the size of the packed files usually grows on average. Those. the BitBuster format is, in a sense, a local optimum. What Pletter does is it slightly generalizes the format of links (only 6-7 variations). The old Pletter 0.4 simply consisted of a universal packer and seven unpackers, which is rather inconvenient. The new Pletter 0.5 (current version 0.5c1) itself tries to pack with each format variation and writes the packed file in the most successful format. The first 3 bits of the file encode the format variation used to configure the unpacker. Even such a small format adaptation significantly increases the average compression, with almost no effect on the decompression speed. Packaging in Pletter 0.5 is done like this:

```
pletter5.exe inputfile [outputfile.plet5]
```

Only one unpacker is included with Pletter 0.5:

1. unpletter5.asm - 170 bytes of non-relocatable code that uses, attention, ALL registers (except IR :)). The decompression speed is 75 clock cycles per decompressed byte.

Pucrunch is one of the very sophisticated old packers, with roots in C64 (here are the source files for the project). I did not understand in detail this packer; I know that this is some rather complicated variation of LZSS, with a reputation for slow unpacking. From the author’s extensive explanations, it’s clear that he definitely did lazy evaluation in his packer. This means that the Pucrunch packer is optimizing; whether the packer is also optimal is unclear. In any case, you can pack the file without the built-in unzipper and additional headers using the slightly hacked version of Pucrunch made by sjasmplus Aprisobal. The command line looks like this:

```
apri_pucrunch.exe -d -c0 inputfile [outputfile.pu_apri]
```

The Aprisobal assembly contains one unpacker:

1. apri-uncrunch-z80fast.asm - takes 255 bytes and decompresses one byte of data per 301 clock cycles (on average).

ZX7 is a very well-developed WoS packer written by Einar Saukas. The official release can be downloaded from WoS . Data packaging is done like this:

```
zx7.exe inputfile [outputfile.zx7]
```

The compression method used in ZX7 is a simple LZSS with standard Elias gamma code for matching line lengths and two link lengths. In a reference to the packer, Einar writes that "the compressed file format is directly based (although slightly improved) on Team Bomba's Bitbuster." “Slightly improved” is translated into Russian as follows:

- ZX7 does not store the decompressed data length (saving two bytes per compressed block);
- Bitbuster kept the length of matches after the offset (this saves one bit per compressed block)
- Bitbuster used inverted Elias Gamma codes (0 <-> 1, I don’t think it would save anything, although it might have simplified the packer a bit).

In short, the ZX7 is just BitBuster, a side view, much like BitBuster is a remake of the old POPCOM packer for CP / M. The only significant difference is that the BitBuster packer is not optimal, but the ZX7 is optimal. Einar writes that its LZSS optimal packing algorithm is new. Of course, we cannot talk about optimal packaging in general - the ZX7 was made in 2013, and, for example, the optimal MegaLZ packer was released in January 2006. I think we are talking about a specific algorithm for optimal packaging. I did not understand the details of his packer; it’s clear that the standard optimal packaging - Dykstra according to the graph of links along the data - will work at best for O (n log (n)) operations, while Einar claims that its algorithm works for O (n).

One significant advantage of the ZX7 is its rather seriously optimized unpackers (the same guys who were involved in optimizing ApLib and Exomizer unpackers helped write them).

1. dzx7_standard.asm - the “standard” unpacker is optimized in length and takes only 69 bytes. Nevertheless, the simplicity of the format allows it to remain fast enough - about 107 cycles per unpacked byte. Used registers: AF, BC, DE and HL;
2. dzx7_turbo.asm - a much faster 88 byte decompressor that decompresses a byte in less than 81 clock cycles;
3. dzx7_mega.asm - optimized for speed decompressor 244 bytes long, 73 cycles per unpacked byte;
4. dzx7_lom_v1.asm - since I actively used ZX7 in my projects, I wrote my own speed-optimized unpacker. Used registers: AF, BC, DE, HL and IX. Accelerations were possible by taking into account the probabilities of conditional transitions; a new unpacker takes 214 bytes and decompresses a byte on average for 69 clock cycles.

Test results

Before starting the test, I packed the data with two leading packers. Rar 4 (a -r -m5) packed my data set up to 559575 bytes; 7-zip (a -r -ms = off) packed the same data up to 550330 bytes. In both cases, pasting files before compression was disabled. These figures can be considered our most likely unattainable ideal. Here is what the results of the packers included in the test showed:

	unpacked	**ApLib**	**Exomizer**	**Hrum**	**Hrust1**	**Hrust2**	LZ4	**MegaLZ**	**Pletter5**	**PuCrunch**	ZX7
Calgary	273526	98192	96248	111380	103148	102742	120843	109519	106650	99041	117658
Canterbury	81941	26609	26968	31767	28441	28791	34976	31338	30247	27792	32268
Graphics	289927	169879	164868	173026	169221	171249	195544	172089	171807	169767	172140
Music	151657	59819	59857	62977	60902	62678	77617	62568	63661	63977	66692
Misc	436944	252334	248220	262508	251890	255363	293542	261396	263432	256278	265121
TOTAL:	1233995	606833	596161	641658	613602	620823	722522	636910	635797	616855	653879

As you can see, only one packer was able to drop below the magic number of 600,000 bytes. The same data in the form of compression ratios (in percent) are as follows:

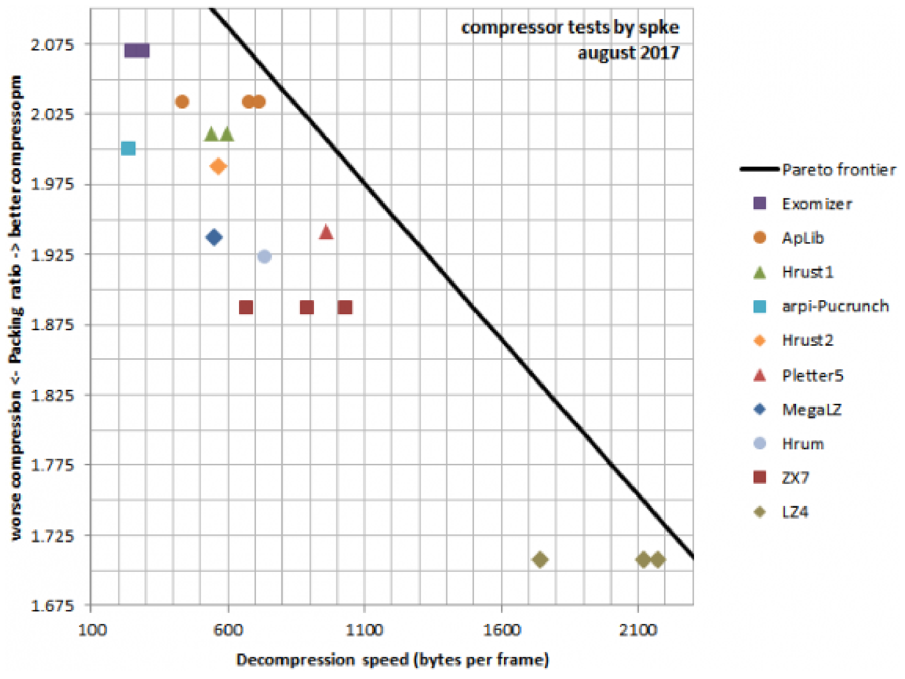
	ApLib	Exomizer	Hrum	Hrust1	Hrust2	LZ4	MegaLZ	Pletter5	PuCrunch	ZX7
Calgary	35.90	35.19	40.72	37.71	37.56	44.18	40.04	38.99	36.21	43.02
Canterbury	32.47	32.91	38.77	34.71	35.14	42.68	38.24	36.91	33.92	39.38
Graphics	58.59	56.87	59.68	58.37	59.07	67.45	59.36	59.26	58.56	59.37
Music	39.44	39.47	41.53	40.16	41.33	51.18	41.26	41.98	42.19	43.98
Misc	57.75	56.81	60.08	57.65	58.44	67.18	59.82	60.29	58.65	60.68
TOTAL:	49.18	48.31	52.00	49.72	50.31	58.55	51.61	51.52	49.99	52.99

As you can see, the Calgary, Canterbury, and Music packages are best used. Graphics packaging turned out to be comparable to packaging of simply mixed or, sometimes, even packed data.

Intuitively, lunch is never free. The better we press the data, the more we will have to calculate when unpacking, the slower our unpacker will be. Therefore, it makes sense to assume that there is some theoretical maximum compression for each decompression speed, or the maximum decompression speed for each compression ratio, limited only by the performance of our computer. The curve that describes this optimal relationship between the compression ratio and the decompression speed will be called the Pareto optimality face. We don’t know where exactly it goes, but we can guess where it is located roughly looking at the performance of existing packers.

What is the point of guessing about the theoretical brink of optimality? In short, it will allow us to understand which packers are most effective in their category. The closer the packer is to the point of optimality, the more efficiently he uses the computing power of the computer to efficiently decompress well-packed data. That is, we will not now measure compression ratios or decompression rates alone. Instead, we will find out where the edge of optimality is located approximately and we will be able to estimate which compressor will most quickly decompress the data with the desired compression ratio or, conversely, which compressor will most densely decompress the data if we need to achieve the decompression speed given in advance.

Based on these considerations, I processed the data above, and combined them with the measurements of the decompression speed, which I presented describing the decompressors (I want to note that the speed was measured only on the “graphics” and “music” data sets). The results of these measurements can be found above, where I gave a detailed description of each packer. The result of this work was the following schedule:



which allows you to do some ...

General observations and conclusions.

- Conditionally, I would divide all the unpackers into several groups, selected based on the complexity of the algorithms and the speed of unpacking closely related to this.
  - The largest compression to date is provided by the Exomizer packer (on average, it will compress your data more than twice). The payback for this is the low decompression speed - about 250 clock cycles per unpacked byte. Judging by the distance to the Pareto optimality margin, packers with a comparable compression ratio should still have a considerable margin in unpacking speed. Unfortunately, Exomizer has no competitors right now.
  - In the second group, I would include “complex” LZ packers: Hrust 1 and 2, Pucrunch and ApLib. ApLib is the clear leader in both compression and decompression speed. Compression ratios of other packers in this group are comparable; Hrust 1 presses slightly better; Hrust 2 presses slightly worse. However, ApLib will usually compress your data 0.5% better than Hrust 1 and decompress it 20% faster using a smaller unpacker (fast, i.e. the Hrust 1 stack unpacker with a length of 209 bytes spends about 120 clock cycles per unpacked byte; ApLib's second fastest unpacker takes 199 bytes and spends about 105 clock cycles per byte). When I need a comparable compression ratio, I personally will use ApLib with one of the fast unpackers.
  - Pucrunch is obviously already obsolete. Hrust 2 clearly loses to Hrust 1 and does not have any obvious advantages, so, apparently, it is also uncompetitive. The wrapper for Hrust 1 is optimal, while all packers for ApLib are optimizing, so the gap in compression can still grow slightly. This leads to the following conclusions: Hrust 1 can occupy a unique niche only if someone writes a new compact unpacker for it, accelerated by one and a half times. When / if someone writes the optimal packer for ApLib, Crunch 1 will probably have to be retired.
  - The third group included a mixture of simpler LZ packers, such as MegaLZ, Hrum, Pletter and ZX7. Earlier, before writing this review, it seemed to me that when you want a little more compression, and you can sacrifice speed - MegaLZ or Hrum will be good candidates. However, now I see that the decompression speed of MegaLZ and Hrum is quite comparable to the decompression speed of Hrust 1 or ApLib, and their compression ratio is much lower. It turns out that MegaLZ and Hrum are already morally obsolete - although if

    their unpackers could be accelerated 1.5-2 times, they would again find use. In almost all of my demos, I sacrificed compression and used ZX7, because in trackmo, decompression speed is usually more important. Nevertheless, looking at the results of Pletter 5, I can not help but conclude that the ZX7 is also outdated,
    - In the fourth group, I again only have one unpacker, LZ4. This is the only byte packer in the review. Given the completely unrealistic speed of decompression - on average, about 34 clock cycles per decompressed byte, i.e. about 1.5 * LDIR, apparently it also makes sense for us on the Spectrum to start thinking about compression, as a way to increase the throughput of the available iron. Just think, LZ4 can decompress more than 2Kb of data per frame!
- The group of encoders that optimized the standard Exomizer unpackers also made an Exomizer packed file repacker, which further accelerates decompression. I did not delve into their work, in my opinion they did something too complicated there, but this once again suggests that the Exomizer unpacker can still be accelerated.
- The compression gap between Exomizer and the subsequent ApPack and Hrust 1 is quite small. I think for applications that require high compression, it would be interesting to see new packers in this range, but in this category of packers it is difficult to wait for significant revolutions, since technologies for more efficient compression (for example, arithmetic coding or PPM) will usually require a significant slowdown and, often, considerable amounts of memory to unclench. See, for example, the development branch of RIP, mRIP, etc.
- Packaging on a PC enabled optimal packaging in many packers. The speed of optimal packers is often low, but the high performance of the PC and the small size of the files relevant for the Spectrum makes optimal packaging, where possible, quite practical.
- The success of Pletter 5, especially in comparison with MegaLZ, shows that instead of re-complicating compressors with specialized codes for some special situations, you can try to make very simple LZ compressors with optimal packaging and dynamic adaptation of the unpacker to specific data. To be honest, Pletter 5 is very primitive; I think that in this area there is great potential for increasing the compression ratio without a significant loss in the decompression speed - it would be nice to push the optimality margin a bit further.
- The gap in compression ratios between byte and mixed bit-byte packers is now very wide, but the experience of fast packers on PC suggests that this gap can be narrowed. Algorithms such as LZF or LZO on PC easily beat LZ4 in compression and can serve as the basis for byte packers of the future, especially if it is good to work on adapting their formats to our limitations and to Z80 features.
- The comparative simplicity of the ZX7 made it the fastest mixed bit-byte packer in the test. Nevertheless, the distance to the Pareto optimality margin of the ZX7 is so great that its non-optimality is obvious. The niche of mixed bit-byte packers, sharpened not for compression, but for quick decompression on the Z80, is, so far, essentially free.
- The simplicity of the ZX7 format allows it to boast the shortest unpacker in this review, which is important for making mini-intras (256b, 512b or 1Kb). Why does the ZX7 manage to deal with much more efficient packers in this case? The difference in compression ratios between the best packer in the review and ZX7 is only 52.99% -48.31% = 4.68%. 5% of 1Kb is 51 bytes. Considering that the compact ZX7 unpacker takes only 69 bytes, it turns out that the Exomizer unpacker should be shorter than 69 + 51 = 120 bytes, so that the Unpacker + Exomizer data takes, on average, less space than the ZX7.

A very short memo for those who do not like letters

1. *“I need the biggest compression, I don’t care how long it will be unpacked then”* => **Exomizer** ;
2. *“It’s important for me to compress better, but I can’t spend more than 110 clock cycles per byte of decompressed data”* => **ApLib** ;
3. *“It is important for me to be able to quickly decompress my data (no more than 75 clock cycles per decompressed byte) without losing too much compression ratio”* => **Pletter 5** ;
4. *“I am only interested in the high decompression speed, compression is just a nice bonus”* => **LZ4** ;
5. *“I really love relocatable unpackers”* => **Hrust 1** with a stack unpacker, **Hrum** if you need a smaller unpacker, **MegaLZ** if it is important to refuse to bind to the stack;
6. *“I make mini-interns (from 256b to 1kb), and the unpacker size is very important for me”* => **ZX7** (69 bytes - standard unpacker, 88 bytes - version of “turbo”). See also ZX7mini, with an unpacker from 39 bytes , which, however, packs data even worse than LZ4.

I understand that it is impossible to see the vast. Of course, I could not look at all the packers on the planet. Some viewed packers seemed uninteresting to me, so I did not include them in the review. A lot of information just disappeared. Therefore, I will be very happy to chat in general about the packers that you use and m. I will even know something about some of them. It will be great if you can share your compression secrets here in the comments, or even know how unpublished before. Good luck compressing your data!

application

Since it was quite painful to search all this information on the Internet, I put together a package of information on packing data for the z80. It includes releases collected on the network and other supporting data about almost every included packer, from which I extracted everything that is required to work with each specific package. All this stuff lies here: compression2017.7z

In addition, I decided to release two of my secret unpackers. The first one is the new unpacker for ZX7, which was previously used in several releases with my participation, including the Break / Space demo. My version of the decompressor is faster than the standard decompressor dzx7_mega.asm by about 4% and takes up 30 bytes less memory. On average, you can expect my unpacker to require about 69 processor cycles for each unpacked byte: dzx7_lom_v1.asm The

second unpacker I wrote for the ZX4 format, collecting all the best ideas from existing unpackers. The new unpacker takes 103 bytes and works 2% faster than the unpacker from Piotr Drapich; for each unpacked byte my unpacker spends about 34 Z80 clock cycles: unlz4_spke.asm

The attached Hrum unpacker is also partly new. Inside it is a standard Hrum unpacker, I just slightly rewrote its interface (and messed up the formatting !!!). It takes 104 bytes and spends 97 cycles to unpack a byte: unhrum_std.asm