Improved Recovery and Reconstruction of DEFLATEd Files

By

Ralf Brown

From the proceedings of

The Digital Forensic Research Conference

DFRWS 2013 USA

Monterey, CA (Aug 4th - 7th)

Digital Investigation 10 (2013) S21–S29

Contents lists available at SciVerse ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

Improved recovery and reconstruction of DEFLATEd files

Ralf D. Brown

Carnegie Mellon University Language Technologies Institute, 5000 Forbes Avenue, Pittsburgh, PA 15213, United States

Keywords:
Data recovery
File reconstruction
ZIP archives
DEFLATE compression
Language models

A B S T R A C T

This paper presents a method for recovering data from files compressed with the DEFLATE algorithm where short segments in the middle of the file have been corrupted, yielding a mix of literal bytes, bytes aligned with literals across the corrupted segment, and co-indexed unknown bytes. An improved reconstruction algorithm based on long byte *n*-grams increases the proportion of reconstructed bytes by an average of 8.9% absolute across the 21 languages of the Europarl corpus compared to previously-published work, and the proportion of unknown bytes correctly reconstructed by an average of 20.9% absolute, while running in one-twelfth the time on average. Combined with the new recovery method, corrupted segments of 128–4096 bytes in the compressed bit-stream result in reconstructed output which differs from the original file by an average of less than twice the number of bytes represented by the corrupted segment. Both new algorithms are implemented in the trainable open-source `ZipRec 1.0` utility program.

1. Introduction

The more than 20-year-old DEFLATE lossless compression algorithm has become ubiquitous, being used in ZIP archives, gzip'ped files, PDFs, and many other applications. Because of its wide-spread use, recovering data from corrupted DEFLATE-compressed files is of significant interest for data recovery and forensic specialists. Programs to extract intact DEFLATE streams from disk images or corrupted archives are numerous, and there are even programs to extract intact DEFLATE packets from a corrupted stream (Brown, 2011; Park et al., 2008). However, none of these can properly deal with corruption *within* a DEFLATE packet.

This paper presents two advances on the state of the art in dealing with corrupted DEFLATE streams. First, we enable the extraction of data from the intact tail end of a DEFLATE packet which contains a relatively short corrupt section (up to 4–8 kilobytes, depending on compression ratio), and show how to realign the history window around the corruption for improved recovery (Fig. 1). Second, we improve the speed and effectiveness of the reconstruction algorithm which infers the values to be assigned to co-indexed unknown bytes in the recovered data. Both improvements are implemented in a new release of the `ZipRec` program.

Sections 4 and 6 present the algorithms for extracting partial DEFLATE packets and re-aligning the history window, while Section 5 presents the improved reconstruction algorithm. Sections 7 and 8 describe the experiments performed on the new algorithms and their results.

For the examples in this paper, the first 2500 lines of held-out English text from the Europarl corpus (Koehn, 2005) are placed in a file and compressed into a ZIP archive containing only that file. The 128 bytes beginning at offset 4096 within the archive (offset 4025 within the DEFLATE stream) are then set to zero. Fig. 3 shows the relevant portion of the original text, beginning at the start of the sentence where the corruption occurs, while the quadrants of Fig. 4 show the results of various methods of extracting data from the corrupted archive, which will be discussed in more detail at various points below.

2. Overview of the DEFLATE algorithm

DEFLATE compression (PKWare Inc, 2012) consists of converting the input into a sequence of intermixed literal

E-mail address: ralf@cs.cmu.edu.

1742-2876/$ – see front matter © 2013 Ralf D. Brown. Published by Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.diin.2013.06.003

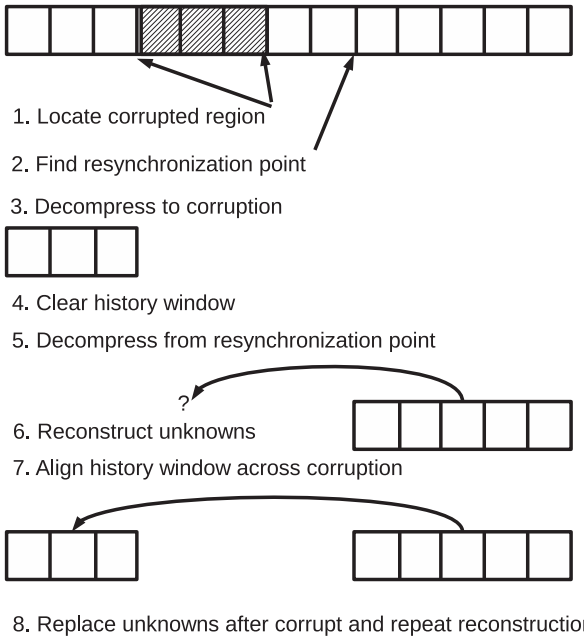

Fig. 1. Extracting data from a DEFLATE stream with a small segment of corruption.

bytes and matches (of length 3–258 bytes) within the prior 32,768 bytes of input, which forms a sliding history window. This LZ77-style (Ziv and Lempel, 1977) redundancy-minimized sequence is then Huffman encoded using a Huffman tree containing all 256 byte values, an End-of-Data symbol, and a number of symbols denoting match lengths. A second Huffman tree encodes match distances; a match-length symbol is always followed by a distance symbol from the second tree. The compressor is permitted to split the output stream into packets at any time it chooses, typically either when its input buffer is full or it determines that a different pair of Huffman trees would improve compression (or that transmitting a section of up to 64 KB as uncompressed bytes would be advantageous). Although the Huffman compression is reset with each new packet, the history window is not, and backward references to matches may span multiple packets if their total uncompressed size is less than the 32 KiB history window. The compressor may *choose* not to emit matches across a packet boundary, which is sometimes done to permit partial extraction after corruption, especially in conjunction with generating a zero-byte uncompressed packet to simplify

```
Bitstream:   1 0 1 0 1 1 0 1 1 1 0 0 1 1 ..
Candidate 1: 1-0
Candidate 2:   0-1-0-1 1-0 1-1-1-0
Candidate 3:     1-0 1-1-0
Candidate 4:       0-1-1 0-1-1 1-0
Resynchronization:                 0 1 1 ..
```

Fig. 2. Resynchronizing DEFLATE bitstream after corruption. Extension of a candidate stops as soon as it matches a symbol boundary of another candidate.

> These people were received and cared for by that town, but quite remarkably, they claimed to have been dropped across the border by the French police. The police had picked them up in Calais, taken them from Calais to the Belgian border and had no qualms about subsequently dropping them off in Belgium. A very strange business, all the more so because, according to other witness statements, it appears that this is not the first time this has happened.
> Fortunately, this incident has been settled at the highest level between the French and Belgian authorities, and it appears that they found a way of discussing it. However, to my great surprise, I was informed by a Belgian that it is not just the French who get up to these tricks, but also the Dutch and Germans. When I asked him if the Belgians do the same thing, he confirmed this and said, now and again. This leads me to conclude that everyone still has the standard European reflex, namely to pass on their problems to their neighbours.
> In my opinion, it is high time that we, as European legislator and as European Parliament, at least pressed for a European reflex. Just as Europe did too little during the oil crisis, it is also making its presence felt insufficiently with regard to this disturbing problem.

Fig. 3. Original text for the extraction example.

finding the start of the next compressed packet (e.g. `zlib`'s `Z_FULL_FLUSH` option (Roelofs and Adler, 2012)). Doing so reduces the compression ratio.

DEFLATE decompression consists of interpreting the bitstream as a sequence of Huffman symbols, and adding a byte to the output if the symbol represents a literal value, or copying the specified number of bytes at the specified distance in the history window to the output. The last 32,768 bytes of output are maintained in a copy of the history window at all times.

The effect of an erroneous symbol in the bitstream is that either an incorrect byte is emitted (if it decodes as a literal), or that an incorrect sequence is copied from the history window to the output (if it decodes as a match length or match distance). In both cases, subsequent back-referenced matches will further propagate the error either by copying the erroneous byte(s) directly or by copying a sequence from an incorrect location due to a difference in the number of bytes output.

3. Related work

Although the DEFLATE algorithm has become ubiquitous, being used in ZIP archives, most XML-based document formats (e.g. OpenDocument, OOXML, and ePUB), and even some network protocols, there has been little work reported on recovering data where the compressed bitstream has been corrupted.

Numerous utilities exist to extract intact members from ZIP archives, e.g. the various ZIP repair programs or Cohen's semantic file carver (Cohen, 2007), but few even attempt to

These people were received and cared for by
that town, but quite remaonal record for me
this autumn!

Yes indeed, Mrs Thors, we shall amend
the Minutes accordingly.
(The Minutes were approved)

Presidency communication on the situa-
tion in the Middle East
Ladies and gentlemen, in the last few ^@-
^@^@^@serious m^@^@^@^@^@em. Pleass-
^@^@^@^@^@^@^@^@^@^@^@^@^@^@-
^@^@^@^@^@^@^@^@^@^@^@^@^@^@e
Comn. Itway of ^@^@^@^@^@^@^@ess.
^@^@^@^@^@^@^@^@^@^@^@^@^@^@-
^@^@^@^@^@^@^@wtemenhItwanday,Fd
woud a way^@^@^@esspoin-Offhave righset-
ndaould ask ainlgradl youe rtweir sorr!

Yes encyproved)
inhore rigThursdn. Itway of ^@^@^@^@-
^@^@euld u Biarritz. Ie the requeomorHg
in theso sychange sur ths all tawith at p is
aayproved)

(a) Standard decompression

These people were received and cared for by
that town, but quite rema??????h????n????F?-
?????????????????????????????????set????????-
?????????????l???????twe?????????????????????????-
???????hor??????????????????????????????e????-
u?????????????????????????????H?????????????y??-
?????sur???s??????a???????????????a???????????-
?????????????????????????????????????g???u??????-
??s???????s????????????????Dut???????Ger???s?-
??????I????e????m?????????????????s???????????-
???h?????????o????m??????????????????,????????-
??????????????l?????????????o??????????????????????-
??????????has???????????rd??????????????lex,???-
??l??????ass???????ir??????????????????ir?n????-
b?????
????????????????????? ??????????????????????????-
?????????l???????o????????????????????????????-
???????????????p?????????????????????????????lex.
J??????????????????????????t????????????????-
????c??sis????????????????????????s?p???????
?????in??????????l????????????????????????????-
urb?????????????????????????????? S???ckx??????-
????no???-
??f????????-
???????????????s??????????????????????????s?

(b) Recovered

These people were received and cared for by
that town, but quite rema

. There People to???received aNe cared ?td by tha?-
to t bow quite remarks????????e conc?ed to have -
been a????????cross ??e??or ??????????C?erch a-
?????????? a??ity o?o pick ????????? in Co??-
ies taken i???????????????????????Br??i?e tl?-
her and ha??????????e aboue subsequently ?r??-
e?e ??????????n Bu???????????????rrange???-
ained ??ill Commuse so because, accord?????????-
??e wolress statement , it appears that this is not a-
???????e tra? this

_??e happened. Fortunately, this incident has bere
settled at the has bst level between the C?erch -
and Br??i?e authoritiesing a it appears that the fo-
und ? way ?f de????s?a???s. However, to my-
great surprise, I was informed by a Br??i?-
e that it is not must the C?erch who go??un ??-
these thicks bow also t?e Dutch and Germans-
When I a?ted him t??t?e Br??i?es w? the same t-
hime ?? con?ram?e this and bert, now and tddin. -
?het l????nger con ond, that everyone still has-
the st??dard European implex, a???ly te passcin-
?their ?????em s?? their nat??b?????
in my opiniot it is has tra? the???e? as European l

(c) Simple Reconstruction

These people were received and cared for
by that town, but quite remarks????????-
e coun?ed to have been a????????Kroes ??-
e bor ??????????C?ench a?????????? a??ite t-
?o pick ????????? in Co? it taken i?????????-
??????????????B?utiae te?her and ha??????-
????s abooe subsequently draws??e ???????-
s in Br???????????????rrange???line t??ill ?-
ommuse so because accord???????????e wolress -
statement , it appears that this is not a???????-
e time this has happened. Fortunately, this incident
has been settled at the highest level between the C?-
ench and B?utiae authorities, and it appears that th-
ey found a way of discussing it. However, to my great
surprise, I was informed by a B?utiae that it is not -
just the C?ench who get up to these tricks, but also
the Dutch and Germans. When I asked him if the
B?utiaes do the same thing, he confirmed this and said,
now and again. This leads me to conclude that everyone
still has the standard European reflex, namely to pass
on their problems to their neighbours. In my opinion,
it is high time that we, as European legislator and as
European Parliament, at least pressed for a European
reflex. Just as Europe did too little during the oil crisis,
it is also making its presence felt insufficiently with
regard to this disturbing problem.

(d) Reconstruction with Realignment

Fig. 4. Comparison of the output after zeroing 128 consecutive bytes at offset 4096 in a zipped English Europarl file. Boldface text was extracted directly, regular was matched across the corruption, underlined was reconstructed, and italicized question marks are unknown bytes.

deal with corrupted members. The Gzip Recovery Toolkit (gzrt) (Renn, 2013) attempts to skip over corrupted portions of a gzip archive by scanning byte-by-byte until the `zlib` (Roelofs and Adler, 2012) decompression code can successfully decompress the stream starting at that point. This works with any file containing DEFLATE streams, but requires the periodic insertion of a `zlib`-style full synchronization point (a zero-length uncompressed packet accompanied by a reset of the compressor's history window) during compression.

Park et al. (2008) recovered partial files using a bit-by-bit scan for decompressable DEFLATE packets. Their Decompression Helper program can recover packets compressed with the fixed, pre-specified Huffman table (which consists of just a three-bit header followed immediately by Huffman-encoded LZ77 symbols) provided that they do not reference any unknown bytes prior to the start of the packet.

This author previously (Brown, 2011) recovered partial files using a faster reverse scan to find the earliest uncorrupted DEFLATE packet in the file, and added a reconstruction method based on word unigrams and byte trigrams. Although there was no restriction on references to unknown bytes when searching for valid packets, most fixed-Huffman packets were skipped due to the large number of false positives when relaxing that restriction. The version of `ZipRec` described in that work required eight to ten times as long to extract a file using its packet scanning as `gzip` or `unzip` require to extract an uncorrupted file. Reconstructing the unknown bytes was time-consuming, taking 200 or more times as long as extraction, e.g. 247 s for a five-megabyte file; it also did not properly support languages using multiple bytes per character, as word-splitting (a prerequisite for the word-based language model) is encoding-specific.

4. Method – recovery after corruption

When corruption occurs in the middle of a packet, the decompressor still has a large amount of valid information available; the most important of which is the two Huffman trees used for entropy-coding the stream of literals and back-references. Given the Huffman trees, it is possible to uniquely segment the bitstream into Huffman symbols starting at any symbol boundary, which in turn permits the sequence of literal bytes and back-referenced matches created by the compressor to be regenerated starting from that symbol. If the decompression starts in the middle of the file, however, the contents of the history window will be unknown, so any bytes which are ultimately copied from a position prior to the start of decompression must be flagged as unknown.

Knowing which range of bytes in the DEFLATE stream are erroneous, the decompressor can simply skip around them, clearing the history window to explicit unknowns before restarting decompression. Unfortunately, it is difficult to detect where corruption has occurred. In fact, unless the decompressor is specifically looking for corruption, it quite likely will not even notice that anything is amiss until decompression is complete and the calculated CRC fails to match the stored value! In early experiments, only four of 100 test runs replacing 512 bytes by random data generated invalid compressed bitstreams. Currently, `ZipRec 1.0` only detects corruption automatically where there is a sequence of at least 128 consecutive identical bytes, the probability of which in an intact file is vanishingly small. On the other hand, file-copy utilities like `dc3dd` or `dcfldd` would generate such a sequence when they replace unreadable sectors by zeros. Multiple attempts have been made to detect corruption in the general case (see Section 9), and a user interface is in development to permit manual marking of corrupted regions.

Following the corrupt region, there is an ambiguity as to where to restart the decompression – any bit boundary is potentially valid, and a Huffman bitstring may be up to 48 bits long. Thus, there are 48 possible restart points, many of which will introduce erroneous bytes into the decompressed byte stream. Since the Huffman codes are designed for an unambiguous segmentation of the resulting bitstream, whenever two candidates segmentations arrive at the same bit position, they will thereafter segment the stream identically to each other. To ensure that no erroneous bytes are introduced, each of the 48 possible starting positions are incrementally decompressed until all of them converge on the same bit boundary (Fig. 2). At this point, any further decompression will be correct. Convergence takes a median of less than 50 bytes and during testing in development never exceeded 150 bytes.

Having found both the start of the corrupt region and resynchronization point after the corruption, recovery consists of decompressing the packet up to the start of the corruption, skipping forward to the resynchronization point and clearing the history window to unknowns, then continuing decompression as normal. To support reconstruction, a marker is left at the point of discontinuity when storing co-indexed bytes instead of plain text. This informs the reconstruction code that the history window needs to be cleared or re-aligned at this point, and once realigned, the marker also stores the computed offset.

Fig. 4(a) shows the output when extracting the file using the Info-ZIP Project's "unzip", version 6.00 (Info-ZIP work group, 2009). The erroneous NUL bytes in the compressed data result in bytes being copied from before the start of the file (which appear as '@') as well as incorrect sequences being copied from other positions and incorrect literal bytes being introduced. For this particular example, the occurrence of references beyond the beginning of the uncompressed data is a clear signal that the bitstream contains errors, but such references would not have occurred if the ZIP archive had been altered at a point further into the file.

Fig. 4(b) shows the same portion of the file when extracted with `ZipRec 1.0`. Because it knows that there was corruption, it can avoid interpreting the bitstream within the corrupt segment and can identify which bytes are the result of indeterminate copies. However, there are now many unknown bytes (though their proportion decreases with distance from the corruption), reducing the usefulness of the output.

5. Method – reconstruction

To increase the usefulness of the recovered data, we can take advantage of the fact that every unknown byte in the recovered output has an associated co-index as a result of (possibly oft-repeated) copying from a single instance prior

to the point at which decompression started. Thus, certain sets of bytes are known to be identical, which combined with contextual clues from language models permits inference of their values.

The reconstruction method in Brown (2011) relies on a unigram word model and a trigram byte model. The method used in this paper eliminates the word model and instead relies entirely on a pair of byte *n*-gram models, with *n* as large as practical (either 7 or 8 for the experiments reported in this paper). Both a forward and a backward (reverse byte order) model are used to permit context scores from both the left and right contexts of an unknown byte. By relying solely on byte *n*-grams, the need for word-splitting is eliminated, and *n*-grams spanning multiple words can help to identify the whitespace and punctuation characters between words, which proved difficult for `ZipRec v0.9` to reconstruct.

Inference of the values for unknown bytes proceeds by repeatedly accumulating scores for each co-index by scanning the buffer containing the decompressed data, determining which co-indices have sufficiently-reliable information to select a replacement, and applying those replacements. Scoring is performed by searching for valid language model *n*-grams to the left and right of an unknown byte, as well as *n*-grams in the forward language model which span the unknown byte, as described below. If both left and right contexts succeed, or a central spanning *n*-gram is found, the context is declared to be good and the score is added to the running total. If only a left context or only a right context is found, it is discarded to avoid hallucinating extensions to existing strings into consecutive unknown bytes based solely on the most likely continuation of the history according to the language model.

The left and right contexts are computed by attempting to match up to $n-1$ bytes to the left of an unknown against the forward language model, and up to $n-1$ bytes to the right of the unknown against the backward language model (see the top half of Fig. 5). Each matching history is then used to predict possible values for the unknown byte in proportion to the frequency of the values as final bytes in the model's *n*-grams. Matches against the language model may be inexact due to unknown bytes, in which case the model score is the average of the probabilities for all matching *n*-grams. To limit computation and avoid uninformative matches, lookups are skipped if the total number of possible matching histories is greater than 72 (based on the set of character values which have not been excluded as possibilities for unknown bytes in the history), and scores are not updated if more than six histories actually match.

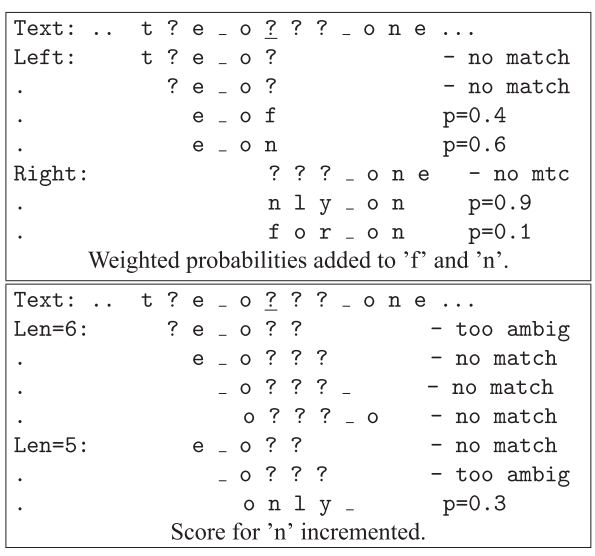

Fig. 5. Left/Right context scoring (top) and Central context scoring (bottom). The underlined question mark is the unknown being scored.

Context matches are attempted from longest (model length less one) down to bigrams (yielding trigram predictions). Once a good match is found, no shorter matches are attempted; longer matches are weighted more heavily. While the code does support using multiple matches of different lengths for a single location, this did not actually improve reconstruction.

Spanning contexts are computed similarly, except that rather than using the conditional probability of matching *n*-grams, the joint probability is used because the byte of interest is no longer the final byte of the *n*-gram. Matches are attempted for all possible positions which include the unknown byte, and again proceed from longest possible down to trigrams. Slightly more ambiguity is permitted in the matches; lookups are only skipped if the total number of possible matches is greater than 240, and the limit on actual matches is increased to eight.

A candidate replacement is considered to be of high reliability if it has at least one good context, and its confidence score is greater than zero and at least 0.97 times the highest confidence score. The confidence score is computed as

$$ratio = min\left(10000, \frac{highest}{second}\right)$$

$$diff = highest - second$$

$$conf = \sqrt{\frac{context}{occur}} \times (\lambda\, log\,(ratio) \times \mu\, log\,(1 + diff))$$

where λ is currently set to 10 and μ to 0.25. This gives a candidate some credit for high absolute scores to avoid the highest overall confidence resulting from a single instance with a high ratio (as could, for instance, happen if there is a single matching spanning *n*-gram in the language model, yielding a second-highest score of zero) and also discounts the confidence for indices which have good contexts for a low percentage of their occurrences.

After the high-reliability replacements (as defined in the previous paragraph) have been applied, processing continues by scoring the remaining unknown bytes with the updated contexts resulting from the replacements. It was found to be advantageous both in speed and overall accuracy to periodically perform a greedy application of replacements. Thus, after the 20th replacement step and every 50th thereafter, all unknown bytes for which the ratio between highest-scoring byte value and second-highest scoring value is at least 25 are replaced by that highest-scoring byte value.

Once no more high-reliability replacements can be found, the iteration terminates. At this point, a final set of replacements is added by selecting the highest-scoring

byte value for each unreplaced co-index, provided that the highest score is at least twice the second-highest score.

To optimize performance, the implementation replaces the sequential scan of the buffer with incremental updates of just the scores affected by a replacement. Each time a set of replacements is selected, an inverted index is consulted to determine their locations, and the *n*-gram scores for all unknowns that could be affected by each instance are reduced by the score they originally received for that instance. The replacements are then made, and the same unknown bytes then have their scores incremented by the language model scores with the new inferred bytes in place.

Fig. 4(c) shows the result of applying the reconstruction algorithm to the example file. The boxed area contains a portion of the reconstructed references which occur within or prior to the corrupted segment. Because no alignment has been attempted, all such back-references are output by the program; the portion shown starts just prior to the start of the example text. One can see that there is a match between the reconstructed bytes and the decompressed text, but it is incomplete and inexact.

6. Method – realigning the history window

Once the text after the corruption has been reconstructed, it is possible to infer the number of uncompressed bytes affected by the corruption. Knowing this number, any back-references in the stream which refer to bytes prior to the corruption can be properly resolved, substantially reducing the number of unknown bytes. A second round of reconstruction is then performed on the remaining unknown bytes. Given the size of the corrupt area in the uncompressed text, the program can also output exactly the correct number of reconstructed bytes to fill in that area, resulting in an output file which is the same size as the original.

To find the proper offset for matches across the corrupt segment, a simple scoring function is applied to each possible offset, and the highest-scoring offset is selected. Unknown bytes are co-indexed by their distance from the start of the decompression; for example, the final three bytes of the history window would have indices 3, 2, and 1, respectively. However, when corruption has caused bytes to be skipped, the indices need to be adjusted by the number of bytes skipped (provided it is less than the size of the history window, 32,768 bytes for standard DEFLATE). If the corruption has caused 1000 uncompressed bytes to be skipped, the final three decompressed bytes before the corruption would have indices 1003, 1002, and 1001.

Determining the correct alignment thus consists of sliding the reconstructed values of the history window against the final 32,768 decompressed bytes and scoring matches. For a given offset, the overall score is a weighted sum of each byte in the overlap region. If either aligned byte is an unknown (the decompressed byte may be unknown for the second and subsequent areas of corruption in a single file), the weight is zero and that position is ignored. Otherwise, the weight is the product of the confidence values for the two bytes (defined as 1.0 if the byte is a literal directly extracted from the DEFLATE stream). The weight is added to the overall score if the two byte values match, and is subtracted if they differ. Fig. 6 shows a simplified version

```
These people were received and cared for by
 The e People wh????ectived ?N? cared ??? b
----------------0000--------0-0-------000--
Net score: -34
```

```
These people were received and cared for by
The e People wh????ectived ?N? cared ??? by
+++-++-+++++++-0000++-+++++0-0+++++++000+++
Net score: +24
```

```
These people were received and cared for by
he e People wh????ectived ?N? cared ??? by
--------------0000--------0-0-------000----
Net score: -34
```

Fig. 6. Re-aligning the history window across an area of corruption by scoring the net matches for each possible shift.

using a fixed weight of 1. Even though there are errors in the reconstruction (differing case is considered a mismatch since the scoring is agnostic about character encodings), the correct alignment still has a dramatically better score than incorrect alignments.

After the history window has been re-aligned, replacements are made for the back-references, including any reconstructed back-references. Literals extracted directly from the bitstream are copied but flagged as aligned rather than directly extracted. Unknown and reconstructed bytes in the history are copied as-is (this will only occur if there are multiple areas of corruption). Replacing unknowns by other unknowns merges their co-indices, improving reconstruction efficacy. Once the appropriate replacements have been made from the aligned history window, reconstruction is repeated to attempt to replace any remaining unknowns. Fig. 4(d) shows the final result. As with Fig. 4(c), the portion of the file immediately following the corrupt region is the most difficult to reconstruct as it has the smallest proportion of literal bytes and a higher proportion of co-indexed unknowns with few occurrences than later sections of the file.

7. Experiments

The data for the experiments reported below consisted of the text of the Europarl corpus (Koehn, 2005), release version 7 (available from http://www.statmt.org/europarl/). The SGML markup on separate lines, which defines document boundaries and metadata, was stripped out. The plain text was then split into training and test files by holding out the October–December 2000 and December 2010 data, and the training and test data were concatenated into one file each for each of the 21 languages of the corpus. Eleven of the languages (Danish, Dutch, English, Finnish, French, German, Greek, Italian, Portugese, Spanish, and Swedish) contain data for the customary October–December 2000 hold-out set as well as December 2010, while the other ten (Bulgarian, Czech, Estonian, Hungarian, Lithuanian, Latvian, Polish, Romanian, Slovak, and Slovenian) were added more recently and only have December 2010 data for testing.

The training data was used to train byte 7-gram language models (8-gram for Bulgarian and Greek, which are predominantly two bytes per character) in both forward and reverse directions for each language for the new

reconstruction algorithm, and word unigram and byte trigram models for the old algorithm. The testing data (both the full file and 2500-line segments with a shorter final segment) was compressed into single-file ZIP archives using the Info-ZIP project's `zip`, version 3.0 (Info-ZIP workgroup, 2008). The test files were then corrupted by replacing a specified portion of the file with zeros or by simulating the same using a command line argument to `ZipRec 1.0`.

Test conditions for recovery included corruption sizes in powers of two from 128 to 4096 bytes. Corruption sizes greater than 4096 were not tested, as 4096 compressed bytes already represent half or more of the 32,768 byte history window for Bulgarian and Greek. For each language, the first 2500-line segment of the training data was used as the original file, and the specified number of bytes starting at offset 4096 within the ZIP archive were set to zero.

The tests comparing the two reconstruction algorithms used a missing beginning to the DEFLATE stream to enable direct comparison with ZipRec version 0.9 (Brown, 2011). The old algorithm was iterated five times to increase recall, extending run-time by 10–15% compared to the default single iteration. Testing used individual archives containing both the full test data for the language as well as each individual 2500-line slice. Although `ZipRec 1.0` incorporates the language identification from Brown (2012) to automatically select the most appropriate reconstruction model, a fixed model was specified on the command line for these experiments.

In evaluating the efficacy of DEFLATE reconstruction algorithms, three measures are of interest: the proportion of unknown bytes for which reconstruction is attempted, the proportion of attempted reconstructions which are in fact correct, and the total proportion of unknown bytes which are correctly reconstructed. The first of these is equivalent to "recall" as used in information retrieval, and affects how much of the output remains unchanged as question marks. The second is closely related to information retrieval "precision", and affects how many erroneously-reconstructed bytes occur in the output. Unlike unknown bytes, these may mislead the user into misreading the content of the file. Finally, the third measure gives an overall measure of the algorithm's effectiveness, being the product of the first two in the same way that F_1 is a product of recall and precision in information retrieval. In simulation mode, `ZipRec` automatically compares the reconstructed file against the original file extracted from the archive without the simulated corruption and outputs statistics on the number of correctly-reconstructed bytes as well as the total number of reconstructed bytes. When the input archive contains actual corruption, only the latter is available without performing a separate comparison step.

8. Results

The two reconstruction algorithms were run separately for each language in the Europarl corpus, using both the full test data and each 2500-line slice, and simulating a missing beginning of the DEFLATE stream (thus forcing decompression to begin at the start of the second packet). Table 1 shows the average percentage of unknown bytes reconstructed by each algorithm for the test files in each language, the average percentage of reconstructed bytes which were correct, and in the final column, the absolute percentage change in correctly reconstructed bytes from the old to the new algorithm (e.g. for Bulgarian, correct reconstruction increased from $0.8255 \times 0.7044 = 58.15\%$ to 95.06%, for a change of $95.06 - 58.15 = +36.91\%$). The final row of the table shows the average across all 21 languages for each column. For all languages, the new algorithm has both a higher reconstruction percentage and a higher percentage of correctly-reconstructed bytes, averaging nearly 8.9% and 14.4% absolute increases, respectively.

Table 1
Percentage of unknown bytes reconstructed (Rec%) and percentage of reconstructed bytes which are correct (Corr%), averaged over all test files for each language, for both old and new algorithms. "Change" is the absolute percentage change in correctly reconstructed bytes (Rec% times Corr%).

Lang	Old algorithm (ZipRec 0.9)		New algorithm (ZipRec 1.0)		Absolute % Change
	Rec%	Corr%	Rec%	Corr%	
bg	82.55	70.44	98.00	97.00	+36.91
cs	83.45	73.30	96.78	95.58	+31.34
da	87.03	86.78	93.09	95.26	+13.15
de	88.35	91.44	96.55	95.84	+11.75
el	91.56	58.26	99.20	99.19	+45.05
en	87.60	89.58	92.72	95.07	+9.67
es	87.35	87.82	92.74	94.92	+11.32
et	84.72	78.60	95.96	95.10	+24.67
fi	85.29	84.23	92.16	93.97	+14.76
fr	87.65	87.91	93.22	95.39	+11.87
hu	86.67	73.82	97.25	96.67	+30.03
it	87.02	89.66	91.08	93.73	+7.34
lt	84.87	76.64	96.43	94.60	+26.18
lv	82.39	73.60	96.42	93.50	+29.52
nl	94.29	90.01	98.48	98.08	+11.72
pl	85.39	78.96	97.10	96.29	+26.07
pt	88.06	85.96	94.00	95.23	+13.82
ro	80.43	78.12	96.43	93.18	+27.02
sk	94.89	86.12	98.69	98.69	+15.68
sl	81.62	78.75	95.81	93.53	+25.34
sv	87.63	84.01	93.53	95.25	+15.46
AVG	86.61	81.14	95.50	95.52	+20.89

Run-time is also considerably shorter for the new algorithm; most test files take ten to twelve times longer with the old algorithm, and a few outliers more than forty times as long. For example, on an Intel i7 processor at 4.3 GHz, the full English test data of 9,745,490 bytes requires 23.6 s for reconstructing 2,000,329 unknown bytes with the new algorithm and 265.6 s with the old one, a factor of 11.25. The full test file for Bulgarian (4,778,360 bytes total, 2,776,241 to be reconstructed) was one of the outlier cases, taking 22.0 versus 1414.6 s, a factor of 64.3.

Table 2
Number of incorrect bytes in reconstructed English output for various sizes of corruption and corresponding number of uncompressed bytes directly affected by the corruption.

Corrupted	Uncompressed	Incorrect	Ratio
128	371	409	1.10
256	706	708	1.01
512	1426	1850	1.30
1024	2591	3755	1.45
2048	5411	7390	1.37
4096	11,217	14,791	1.32

Table 3
Number of incorrect bytes in reconstructed Bulgarian output for various sizes of corruption and corresponding number of uncompressed bytes directly affected by the corruption.

Corrupted	Uncompressed	Incorrect	Ratio
128	763	1632	2.14
256	1124	2051	1.82
512	2048	10,345	5.05
1024	4421	14,056	3.18
2048	8398	20,774	2.47
4096	16,671	34,631	2.08

Next, we examine the performance of the new reconstruction algorithm coupled with partial-packet recovery and history window realignment. Tables 2 and 3 show how many incorrect bytes remain after reconstruction given a specified amount of corruption of the compressed bitstream for English and Bulgarian. The latter is the language with the poorest overall performance on this particular task.

Each row of the two tables shows the number of bytes corrupted, the number of uncompressed bytes represented by that corruption plus the necessary resynchronization of the bitstream thereafter, the number of bytes which differ between the original file and the reconstruction, and the ratio of the previous two numbers (smaller is better). A ratio of less than 1.0 indicates that `ZipRec 1.0` reconstructed more bytes in the corrupted segment correctly than it made errors on subsequent unknown bytes. This desirable outcome was achieved in six cases: 128-byte corruption for Latvian (498 bytes affected versus 485 bytes incorrectly reconstructed), 128 and 256 corrupted bytes in Polish (304 versus 288 and 622 versus 560 bytes, respectively), and 128, 256, and 512 corrupted bytes in Hungarian (346 versus 235, 733 versus 540, and 1397 versus 1383 bytes, respectively).

Table 4 presents the mean and median values for the ratio of output errors to corruption across all 21 languages of the Europarl corpus. As can be seen, there is no systematic variation in the ratio with increasing amounts of corruption.

9. Ongoing and future work

A user interface to permit manual marking of corrupted regions and manual entry or correction of reconstructed byte values is in development. This will permit files in which corrupt segments currently can not be detected automatically to be processed and to improve the accuracy of reconstruction.

Two unsuccessful attempts have been made to determine corruption points automatically in the general case, and a third method which showed promise in initial experiments is being finalized as of this writing. The first unsuccessful attempt used the same *n*-gram based language identification code which is also used to automatically select a reconstruction language model, declaring corruption to be present when there is a sharp drop in the maximum score for any language. The second attempt used a word-length model. While both methods showed a drop in scores at the point of corruption, the natural variation in scores is larger than the drop due to corruption. The third method uses a word unigram model, declaring corruption when the percentage of words not in the model exceeds one-third for a 512-byte decompressed block. This will naturally produce an erroneous report of corruption if the file should contain text in multiple languages, so dynamic selection of the model based on language identification will be explored. A second draw-back of the word-based approach is that different character encodings will require different word-segmentation code, as was also noted with the unigram word models used for reconstruction in `ZipRec v0.9` (Brown, 2011).

Table 4
Ratio of incorrect output bytes to corrupted uncompressed bytes for a given number of corrupted compressed bytes, averaged across all 21 Europarl languages.

Corrupted	Mean	Median
128	1.542	1.461
256	1.439	1.355
512	1.652	1.473
1024	1.559	1.475
2048	1.544	1.473
4096	1.499	1.465

We also attempted to infer the Huffman trees used by a DEFLATE packet missing its beginning (and thus the transmitted trees) by performing a reverse search from the end of the packet, tracking the set of possible Huffman trees which could generate the bitstream from that point to the end of the packet. This proved to be infeasible when processing the packet in isolation, but combining the search with constraints set by reconstructed back-references may lead to success. This would substantially increase the amount of data recoverable from files for which the beginning is missing or corrupted.

The current approach of finding the point at which all possible segmentations of the bitstream resynchronize after the end of the detected corrupt region causes some otherwise useful bytes to be discarded. Repeating the resynchronization scan in light of reconstructed byte values may permit the appropriate bit offset among the 48 possible starting positions to be determined, limiting the loss to a maximum of six bytes instead of the current median of nearly 50.

While the program code supports one area of corruption in each DEFLATE packet of a file, performance with multiple areas of corruption in a single file has not yet been evaluated.

Finally, the reconstruction algorithm could be strengthened with regard to XML-style markup tags. Experience has shown that such tags are often the most difficult to reconstruct precisely *because* of their highly-repetitive nature. The interior characters of a tag may be represented by a sequence of co-indices which occur in no other contexts (especially if the tag is all in uppercase), leaving the reconstruction algorithm without an opportunity to make inferences. Detecting such single-context sequences and giving the language model matching more leeway to span unknown bytes in such cases, together with longer *n*-grams for tags in the language model, may improve the reconstruction of markup.

10. Conclusions

We have presented a method for recovering data from DEFLATE streams with small areas of corruption which,

together with an improved and faster reconstruction algorithm, yields a reconstructed file which differs from the original input to the compressor by only slightly more bytes than are represented by the corrupted segment of the DEFLATE stream; in six of the 126 test cases, the output differed by *fewer* bytes. Although the method of recovering the sequence of literal bytes and co-indexed unknown bytes is specific to the DEFLATE algorithm, the reconstruction algorithm is applicable with minimal adaptation to any LZ77-based (Ziv and Lempel, 1977) compression algorithm, provided that the compressed bitstream can be converted into such a sequence.

The code implementing the new methods is available at http://ziprec.sourceforge.net/ under the terms of the GNU General Public License.

References

Brown Ralf D. Reconstructing corrupt DEFLATEd files. Digital Investigation 2011;8:S125–31. (Proceedings of the eleventh annual DFRWS conference, New Orleans, August 1–3, 2011).

Brown Ralf D. Finding and identifying text in 900+ languages. Digital Investigation 2012;9:S34–43. (Proceedings of the twelfth annual DFRWS conference, Washington DC, August 6–8, 2012).

Cohen MI. Advanced carving techniques. Digital Investigation September–December 2007;4(3–4):119–28.

Info-ZIP workgroup. Zip, version 3.0. http://info-zip.org/Zip.html; July 2008 [accessed 29.04.13].

Info-ZIP workgroup. UnZip, version 6.0. http://info-zip.org/UnZip.html; April 2009 [accessed 29.04.13].

Koehn Philipp. Europarl: a parallel corpus for statistical machine translation. In: Proceedings of the tenth Machine Translation Summit (MT Summit X). Phuket: Thailand; 2005. p. 79–86.

Park Bora, Savoldi Antonio, Gubian Paolo, Park Jungheum, Hee Lee Seok, Lee Sangjin. Data extraction from damage compressed file for computer forensic purposes. International Journal of Hybrid Information Technology October 2008;1(4):89–102.

PKWare, Inc. Application note on the .ZIP file format, version 6.3.3. http://www.pkware.com/documents/casestudies/APPNOTE.TXT; September 2012 [accessed 29.04.13].

Renn Aaron M. The gzip Recovery Toolkit. http://www.urbanophile.com/arenn/coding/gzrt/gzrt.html; 2013 [accessed 18.02.13].

Roelofs Greg, Adler Mark. zlib home page. http://zlib.net/; 2012 [accessed 18.02.13].

Ziv Jacob, Lempel Abraham. A universal algorithm for sequential data compression. IEEE Transactions on Information Theory May 1977; 23(3):337–43.