COMPACT PRESERVATION OF SCRAMBLED CD-ROM DATA

Jacob Hauenstein

Computer Science Department,
The University of Alabama in Huntsville, Huntsville, Alabama, USA

ABSTRACT

When preserving CD-ROM discs, data sectors are often read in a so-called "scrambled mode" in order to preserve as much data as possible. This scrambled data is later unscrambled and further processed into a standard CD-ROM disc image. The process of converting the scrambled data into a standard CD-ROM disc image is potentially lossy, but standard CD-ROM disc images exhibit much higher software compatibility and have greater usability compared to the scrambled data from which they are derived. Consequently, for preservation purposes, it is often necessary to store both the scrambled data and the corresponding standard disc image, resulting in greatly increased storage demands compared to storing just one or the other. Here, a method that enables compact storage of scrambled data alongside the corresponding (unscrambled) standard CD-ROM disc image is introduced. The method produces a compact representation of the scrambled data that is derived from the unscrambled disc image. The method allows for (1) storage of the standard unscrambled disc image in unmodified form, (2) easy reconstruction of the scrambled data as needed, and (3) a substantial space savings (in the typical case) compared to storing the scrambled data using standard data compression techniques.

KEYWORDS

Compact disc, compression, data preservation, scrambled

1. INTRODUCTION

In recent years, there has been increased interest in preserving digital data, and there has been especially strong interest in the preservation of video games-related data [1–5]. In the case of data stored on a physical medium (e.g., floppy disk, magnetic tape, or optical disc), the preservation process typically consists of extracting data from the aging physical medium (a process called *dumping* or *imaging*) and storing the resulting data (often called a *dump*, *image*, or *disc image*) on modern digital storage devices. A great deal of digital preservation work has historically been accomplished through community efforts [2] whereby a community of users works to dump and preserve physical media that is near the end of its expected life or otherwise believed to require preservation. Because the goal of preservation is to preserve the dumps over a long period of time, such dumps are typically stored with redundancy (i.e., on multiple different modern media and/or in multiple physical locations) in order to safeguard the data. As such, the data storage requirements for preservation communities may grow very large, especially when dumping media that stores large amounts of data, such as compact disc read-only memory discs (often denoted CD-ROMs or simply CDs), the preservation of which is the focus of this work.

Dumps are typically stored in a standard file format. The specific standard format used is decided upon by the community. Usage of a standard format guarantees that all community member's dumps are in the same format, ensuring high software compatibility for each dump and enabling easy comparison between dumps from different community members via standard file hashing

DOI: 10.5121/ijcsit.2022.14401

algorithms. In some cases, the process of dumping a medium produces two sets of data: the final dump in the standard format, and the intermediate data that is processed into the final dump. Unlike the final dump, the intermediate data is often in a format that has relatively limited software compatibility and may be difficult to compare between community members. However, both the final dump and the intermediate data have potential importance in preservation. While the final dump is important because it allows easy comparison of dumps between community members and has wide software compatibility (e.g., with emulators or disc image processing software that enables exploration and study of the data), the intermediate data is important because it may contain data that, due to limitations of the standard used for final dumps, is not included in the final dump. E.g., in Section 2.3, we describe in detail how data may be lost when CD-ROM dumps are processed from the often-used intermediate *scrambled* data into the standard *unscrambled* disc image used for final dumps. Thus, for the case of CD-ROM dumps, there is a need for community members to store both intermediate data and final dumps, imposing even greater storage requirements on top of the already demanding storage requirements of CD-ROM preservation.

The primary contributions of this work are (1) a novel method for compactly storing the intermediate scrambled data alongside the final dump when preserving CD-ROM discs, and (2) a study of the space savings afforded by our method compared to naively storing the intermediate scrambled data. By compactly representing the scrambled data, our method can help ease the storage requirements of the CD-ROM preservation community. Our method exploits that fact that, because the intermediate data can typically be *almost* exactly reproduced during this process, the resulting binary diff between the reconstructed and original intermediate data is often substantially smaller than the original intermediate data (while still allowing recreation of the exact intermediate data). Thus, with our method, a substantial space savings can potentially be achieved while still preserving both the final dump and the intermediate data.

The remainder of this work is organized as follows. Section 2 provides requisite details about data storage on CD-ROM discs and in CD-ROM disc images, including details about scrambling and why some data may not be preserved when the final dump is built from the scrambled intermediate data. Section 2 also presents some details about how scrambled data is dumped from CD-ROM discs, and why it is valuable to do so. Section 3 describes our method for compactly preserving the scrambled data alongside the unscrambled final dump. Section 4 describes the experiments performed to analyze the space savings of our method and the results of those experiments. Section 5 concludes the work.

2. BACKGROUND

This section presents necessary background details about how data is stored on CD-ROMs, why and how such data is scrambled, why there is value in dumping / preserving the scrambled data, and why it may be the case that there is data present in the (intermediate) scrambled data that is removed when a dump is converted from its scrambled form into a standard (unscrambled) image file (i.e., the final dump).

2.1. Data Storage on CDs / Disc Images

In this section, we present some necessary background details about how information is stored on CDs and in CD disc images / dumps. Note that, because the CD specifications are quite lengthy and complex (e.g., as partially seen in [6]), we present here only enough details to aid understanding of this work. Additionally, our focus here is on the way that CDs are presented at the software level when such discs are read by standard, widely available computer optical disc

drives (such as those used for dumping CDs). CDs also contain a large amount of other data (e.g., [7], [8]) at the physical layer that is not exposed / accessible at the software level by such drives and is thus outside the scope of this work and not discussed here. Since our focus here is on preservation, we also assume that discs will be read in a mode that returns the most data possible from the disc.

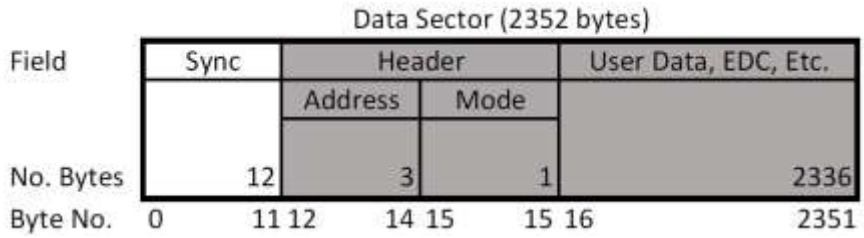


Figure 1: The layout of a data sector. The fields with a white background are not scrambled, and the fields with a gray background are scrambled. The specific layout for bytes 16 to 2351 depends on the mode of the sector.

There exist reading modes that discard some error detection and correction data when reading from discs [9], but we assume those reading modes are not being used here.

Compact discs are divided into *sectors*, and each sector contains 2352 bytes. (N.B., there are some additional bytes present in the so-called *subchannels*, but we do not make use of these subchannels in this work.) When the contents of a CD are dumped and stored in a standard disc image, the disc image simply contains the 2352 bytes of every sector present on the corresponding CD (starting with the first sector). Thus, the logical format of CD sectors presented in this section also applies to CD disc images. (N.B., CD disc images often contain, in addition to the file that holds the sector data, other files that are used to store metadata, but we do not make use of these other files / data in this work.)

Originally, CDs were designed for storage of stereo audio at a 16-bit sampling resolution and a 44.1Khz sampling rate, and thus 75 sectors represents 1 second of audio [9]. Later, the Yellow Book CD-ROM standard was developed for data storage on CDs [9]. As per the Yellow Book standard, when storing data within a sector, the sector is divided into a number of fields. The first 12 bytes (bytes 0 through 11) of the sector are used to store the *sync field* value of 00 FF FF FF FF FF FF FF FF FF FF 00 hexadecimal. The next 3 bytes (bytes 12 through 14) are used to store the *sector address*, and the next byte (byte 15) is used to store the *mode* [6]. Collectively, the 4 bytes making up the sector address and mode are known as the *header field*. The usage of the remaining bytes within the sector depends on the mode chosen for the sector. In mode 1, there are 2048 bytes of *user data* (i.e., the actual user data stored within the sector), and the other remaining bytes are used to store error detection (EDC) and error correction (ECC) data for the sector (as well as some bytes of zero fill data) [6,9]. In mode 2, there are several sector forms, and a sector can hold 2048, 2324, or 2336 bytes of user data depending on the form (with varying amounts of EDC/ECC data). An illustration of the data sector layout is shown in Fig. 1

The presence of a regular bit pattern (i.e., many more bits with a value of zero than one, or vice versa) on the physical disc is problematic for the CD decoding hardware within the optical disc drive [6]. Because such sequences may naturally occur in data, before each data sector is stored on the disc surface, the data sector is subjected to a process known as *scrambling*. In the scrambling process, each byte within the sector, except the 12 bytes that comprise the sync field, is XORed with the corresponding byte in a standardized scrambling table. (In Fig. 1, the fields with a white background remain unscrambled, and the fields with a dark background are

scrambled.) The byte values contained in the scrambling table are designed to, when XORed with the sector data, avoid any problematic bit patterns. The algorithm used to generate the scrambling table is standardized and described in various standards documents (e.g., [6]). Because this scrambling process is XORbased, it is easily reversible by simply performing the same XOR a second time. Thus, data is easily scrambled before the sector is written to the disc (to avoid the problematic bit patterns) and unscrambled when the sector is read from the disc (to return the data to its original state).

2.2. Reading Scrambled Data

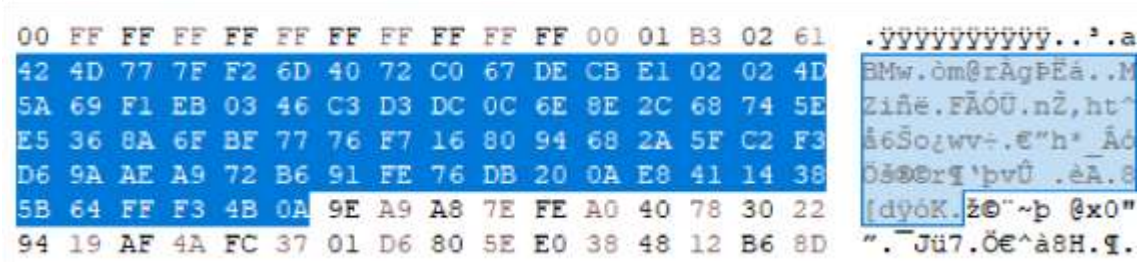
When a sector is read from a CD using a standard optical drive, a sequence of 2352 bytes is returned by the drive. If the optical drive is instructed to read the sector in data mode, the optical drive (typically) automatically unscrambles the data and performs error detection and correction using any EDC / ECC bytes present within the data sector. The drive then returns a sequence of 2352 bytes representing the unscrambled, error-corrected sector starting at the beginning of the sector (i.e., the 12 byte sync field). In contrast, if the drive is instructed to read the sector in audio mode, the drive does not attempt to perform unscrambling (because audio sectors are not scrambled) or use any EDC / ECC bytes within the sector (because audio sectors do not contain EDC / ECC bytes) prior to returning a sequence of 2352 bytes representing the audio sector. When reading the sector in audio mode, the sequence of bytes returned by the optical drive typically does not begin exactly at the start of the sector. Instead, the data returned by the drive is offset by some number of bytes from the true start of the sector, and the offset amount depends on the specific optical drive model used. This audio offset has been studied widely within the optical disc preservation community (e.g., [10], [11]). In addition to this audio offset exhibited by the specific optical disc drive used, some otherwise identical discs exhibit different audio offsets (i.e., the discs contain the same data offset by different amounts) due to variations in manufacturing (often called the *factory offset* or *write offset*) [10]. These offsets complicate the process of preserving discs and comparing dumps between different community members / different copies of a disc, especially for discs containing both data and audio sectors (where it may be necessary to manually adjust for the difference in offsets between the two types of sectors [12]).

In general, optical disc drives refuse to read audio sectors in data mode (or vice versa), as this is the behavior required by the standard optical drive reading commands [9]. However, some optical drives are able to read both data and audio sectors in audio mode [13], bypassing the drive's data sector processing logic. This ability to read data sectors in audio mode (sometimes called *scrambled mode* [14]) is useful for multiple reasons. First, it ensures that the drive returns both audio and data sectors using the same offset, obviating the need for users to manually adjust for the offset difference between audio and data sectors. Second, it bypasses the optical drive's data unscrambling and error correction logic. This bypass is useful because some discs contain data sectors with intentional EDC/ECC errors (often called *error sectors*) as a form of copy protection [15], and bypassing the optical drive's error correction logic allows these error sectors to be processed in software with minimal interference from the drive's error correction logic. The community-developed DiscImageCreator software [14] uses scrambled mode for CD dumping and is capable of automatically correcting for offsets and dumping CDs containing a wide variety of copy protection schemes.

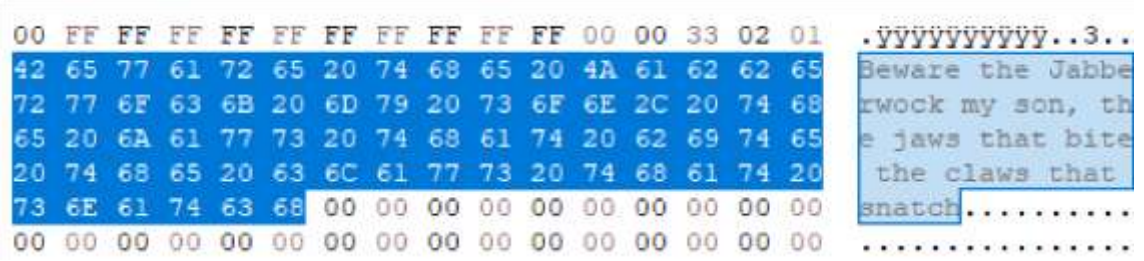
2.3. Converting from Scrambled Data to the Final Dump

While reading data in scrambled mode is useful for CD preservation, the scrambled mode data has relatively limited software compatibility and, because the optical disc drive does not use any error correction to verify / correct errors when data sectors are read in scrambled mode, the

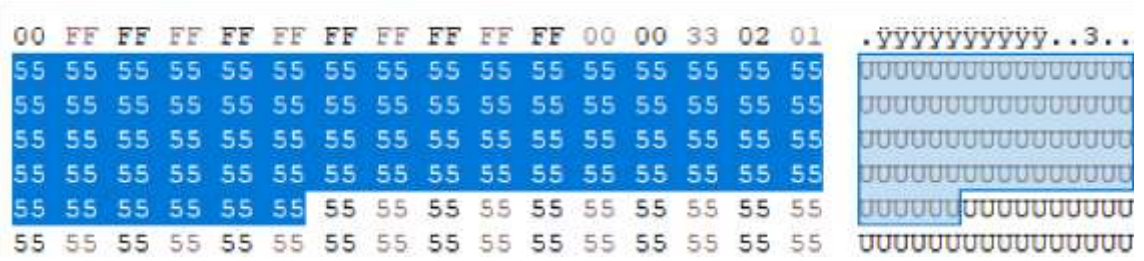
scrambled data may contain undetected errors. Consequently, the community-dictated standards typically used for preserving and comparing CD disc images require that the data sectors be further processed and stored in unscrambled form for the final dump. Thus, the scrambled mode data is an intermediate format. Building the final dump requires that the scrambled data be unscrambled. In addition, any EDC/ECC data is verified during build of the final dump. According to the community standards, for any sectors containing EDC/ECC errors (intentional or otherwise), all bytes



(a) Scrambled



(b) Unscrambled



(c) Final dump

Figure 2: A portion of a sector from the game Rune [16] as seen in a hex editor (left shows raw byte values, right shows text given by those bytes). The highlighted portion of the scrambled intermediate data of the sector (shown in (a)) contains a string of text that can be seen clearly when the sector is unscrambled (shown in (b)). But, because the sector contains intentional EDC/ECC errors, the string of text is removed and replaced with dummy values when the sector is converted into the final dump (shown in (c)).

except the sync field and header field are replaced with the hex sequence 0x55 [17]. This dummy sector standard has a number of benefits for the community, including (1) it matches the behavior of software previously used for preservation of optical discs [12], ensuring that dumps made with newer software match those dumps made with older software, (2) it makes it possible to easily match dumps between users even in the case of discs with intentional errors by making the byte

values in error sectors consistent between dumps, and (3) it was previously assumed that error sectors do not contain any useful data [18], and it was thus believed to be the case that there is no harm in replacing the data in such sectors.

The assumption that error sectors do not contain any useful data has been found to be incorrect for some discs [18]. For example, some CD-ROM copies of the PC video game Rune [16] have hidden text data stored in at least one error sector [18]. This hidden text string is present (in scrambled form) in the intermediate data, but it is destroyed when the intermediate data is processed into the final dump, as shown in Fig. 2. Thus, in order to preserve as much data as possible from each dumped disc, and to ensure that each dumped disc is stored in a standard format, it is necessary for community members to store *both* the intermediate data and the final dump. The intermediate data and the final dump are each equal to the total size of the disc being dumped (i.e., they both contain all the sectors on the disc), and the need to preserve both the unscrambled data and the final dump thus essentially doubles the data storage requirements for each CD-ROM disc dumped (compared to storing only the final dump).

3. METHOD

In this section, we introduce our method to compactly store the intermediate scrambled data alongside the final dump when preserving CD-ROM discs. To ensure that the convenience of the final dump is not lessened when our method is applied, our method leaves the final dump unmodified and converts the intermediate scrambled data to a more compact form. This compact form can easily be used to fully reconstruct the original intermediate data.

Our method takes advantage of the fact that, for data sectors in which no EDC/ECC errors are present, the unscrambling process is exactly reversible. That is, such sectors can be rescrambled from the final dump into a byte sequence identical to the corresponding sector in the intermediate data. In contrast, for sectors in which EDC/ECC errors are present, the sectors are replaced with dummy sectors in the final dump (as noted in Section 2.3), and, consequently, the intermediate data for these sectors cannot be reconstructed by rescrambling the final dump. Thus, to preserve the intermediate data alongside the final dump, our method's compact representation of the intermediate scrambled data stores the intermediate data only for those sectors that cannot be exactly reconstructed from the final dump (i.e., sectors with EDC/ECC errors). Because it is typically the case that the vast majority of data sectors on a CD-ROM do not have any EDC/ECC errors, our method assumes that most data sectors can be exactly reconstructed into their intermediate format. (Our method also works for discs with a large number of EDC/ECC errors, though the space savings will be reduced.)

In the following two subsections, we describe how our method creates the compact representation of the intermediate data and how our method recreates the intermediate data from this compact representation, respectively.

3.1. Creating the Compact Representation

To create a compact representation of the intermediate data from the final dump, our method works in two phases. The first phase creates an approximate reconstruction of the intermediate scrambled data from the final dump. For convenience, we use ϵ to denote the file containing the original intermediate data produced during the dump and $\hat{\epsilon}$ to denote the file containing the approximately

Algorithm 1: Creating Δ from ω via $\epsilon\hat{}$

```
Data: T
Input: ϵ, ω Output:
ϵ̂, Δ
// Phase 1: construct ϵ̂ from ω
        foreach 2352 byte sector s in ω
        do if first 12 bytes of s equal
  then  sync field value
            // data sector, XOR with T
            for i ← 12 to 2351 do
                // XOR byte i of s with byte i −
            12 of T  s[i] = s[i] ⊕ T[i − 12]
            end
            // copy scrambled s into ϵ̂
            copy s into ϵ̂
  else      // audio sector, just
            copy into ϵ̂ copy s into ϵ̂
            end
  end       // Phase 2: now that ϵ̂ is constructed, use xdelta3 to
            diff ϵ̂ and ϵ, giving Δ
Δ ← output of "xdelta3 -e -9 -s ϵϵ̂ "
```

reconstructed intermediate data. For this phase, the input to our method is the disc image file containing the final dump, denoted ω, and the output is $\epsilon\hat{}$. This first phase works as follows. For each sector in ω, the sector is first checked to see if the first 12 bytes of the sector contain the sync field value. If the sync field value is not present, the sector is assumed to be an audio sector, and the sector is copied unmodified into $\epsilon\hat{}$. If the sync field value is present, each byte in the sector (excluding the 12 bytes in the sync field) is XORed with the corresponding byte in a table of the 2340 scrambling values (denoted **T**) and then written into $\epsilon\hat{}$. (Note that, because the first 12 bytes of the sector are not scrambled, the 13th byte of the sector is the first byte that is scrambled, and it is scrambled by XORing with the 1st byte of **T**.) This scrambling table is generated from the algorithm given in [6]. This process is performed for each sector present in ω. Upon conclusion of the first phase, $\epsilon\hat{}$ contains an approximate reconstruction of the intermediate data.

The second phase uses the *xdelta3* binary diff software [19] to encode the differences between $\epsilon$and $\epsilon\hat{}$into a new diff file, denoted Δ. For this phase, the inputs to our method are ϵ and $\epsilon\hat{}$, and the output is Δ. Because, in phase 1, most sectors are exactly reconstructed from the final dump into their intermediate form, ϵ and $\epsilon\hat{}$ typically differ in relatively few byte positions (as few as 0 byte positions may differ), and, as a result, Δ is typically substantially smaller than ϵ. And, because the output of *xdelta3* is Δ, a binary diff file that can be used to reconstruct ϵ from $\epsilon\hat{}$, and, because $\epsilon\hat{}$ can be reconstructed from ω, just the binary diff file Δ is sufficient to reconstruct ϵ from ω. Thus, ϵ can be discarded and the smaller Δ kept instead. Note that this approach to encoding ϵ from $\epsilon\hat{}$ is robust, because, even if some of the sectors were processed incorrectly when $\epsilon\hat{}$ was created in phase 1, Δ still contains the necessary information to rebuild ϵ from $\epsilon\hat{}$. That is, as long as phase 1 results in a $\epsilon\hat{}$ that *approximately* reconstructs ϵ at most byte positions, Δ will be smaller than ϵ.
(We study the amount of space savings achieved by our method in Section 4.)

The pseudocode for both these phases is shown in Algorithm 1.

4.1. Recreating the Intermediate Data from the Compact Representation

Algorithm 2: Recreating ϵ from ω via $\hat{\epsilon}$ using Δ

```
Data: T
Input: Δ, ω Output:
ϵ̂, ϵ
// Phase 1: construct ϵ̂ from ω
        foreach 2352 byte sector s in ω
        do if first 12 bytes of s equal
  then  sync field value
            // data sector, XOR with T
            for i ← 12 to 2351 do
                // XOR byte i of s with byte i −
            12 of T  s[i] = s[i] ⊕ T[i − 12]
            end
            // copy scrambled s into ϵ̂
            copy s into ϵ̂
  else      // audio sector, just
        copy into ϵ̂ copy s into ϵ̂
        end
  end   // Phase 2: now that ϵ̂ is constructed, use xdelta3 to
        apply Δ to ϵ̂, giving ϵ
ϵ← output of "xdelta3 -d -s ϵ̂ Δ"
```

To reconstruct the intermediate data from the final dump, our method again works in two phases. The first constructs $\hat{\epsilon}$ from ω, and this phase is identical to the first phase described in the previous section. The second phase uses *xdelta3* to reconstruct ϵ using Δ. For this phase, the inputs to our method are $\hat{\epsilon}$, the approximate reconstruction of ϵ obtained in phase 1, and Δ.

The pseudocode for both these phases is shown in Algorithm 2.

5. EXPERIMENTS AND RESULTS

In this section, we describe our experiments to evaluate the space savings of our method compared to storing the intermediate data using a standard data compression algorithm and the results of those experiments.

5.1. Experiments

To study the space savings of our method, we first selected and dumped 10 CD-ROM discs using DiscImageCreator. As previously mentioned, DiscImageCreator dumps CD-ROMs using scrambled mode and produces both scrambled intermediate data and a final unscrambled dump. For each disc, we applied our method to the scrambled data, producing a Δ file for each disc. To make it possible to compare the space savings of our method against the space savings afforded by a standard data compression algorithm, we also, for each disc, created a 7-Zip archive of the scrambled data using 7-Zip's "Ultra" mode [20]. Finally, we, for each disc, compared the file size of the compact representation generated by our method and the file size of 7-Zip compressed version of the scrambled data against the file size of the original scrambled data. To evaluate the

space savings, the *space saving* measure, denoted k, was calculated using the compressed file size (i.e.,

Table 1: Details of the 10 discs used in our experiments, including the total number of sectors, number of sectors containing intentional errors, and whether or not the disc contains any audio sectors.

Disc	No. Sectors	No. Err. Sec.	Audio Sectors?
D1	209,671	0	No
D2	319,909	0	Yes
D3	335,411	585	No
D4	353,241	583	No
D5	184,917	2	Yes
D6	308,154	0	Yes
D7	322,560	0	No
D8	282,394	0	Yes
D9	310,667	0	No
D10	147,847	581	No

the size of the 7-Zip compressed file or Δ) and the original file size (i.e., the size of the original scrambled data) according to

$$k = 1 - \frac{\text{Compressed File Size}}{\text{Original File Size}}. \quad (1)$$

The behavior of k is such that it is equal to 0 if the compressed file size and original file size are equal (i.e., when there is no space savings), and it achieves a maximum value of 1 in the case where the compressed file size is 0 bytes.

The 10 discs dumped, denoted **D1** through **D10**, were selected such that they represent a variety of possible disc types that may be input to our method. Some discs contain data sectors only and do not contain any intentional error sectors. Some discs contain both data sectors and audio sectors but do not contain any intentional error sectors. Finally, some discs contain data sectors only and contain intentional error sectors. Details about each of the 10 discs are given in Table 1.

5.2. Results

The results are summarized in Table 2. There, the original size for each disc is shown as well as the size of the 7-Zip compressed scrambled data, the size of Δ, and the space savings value k is shown for both 7-Zip and our method.

For this set of discs, our method achieves a much larger space savings on all discs compared to 7-Zip. Unsurprisingly, our method achieves the highest level of space savings on discs that do not contain any error sectors. This is because, in the absence of error sectors, the intermediate data can (typically) be exactly reconstructed from the final dump.

We believe the superior space savings exhibited by our method in these experiments is primarily due to two factors. First, because the scrambling process tries to avoid regular bit patterns within each sector, the scrambled intermediate data typically has a high level of entropy that makes it

difficult to compress with standard data compression algorithms. Second, unlike standard data compression methods that are unaware of the scrambling process, our method is able to exploit the similarity between the final dump and the scrambled intermediate data using domain-specific knowledge about the scrambling process.

6. CONCLUSION

In this work, we introduced a new method for compactly storing intermediate scrambled CD-ROM data alongside final CD-ROM dumps. Our method takes advantage of the fact that it is possible

Table 2: Results of our method and 7-Zip Ultra compression on the 10 discs. All sizes in bytes. Largest *k* value for each disc is in **bold**.

Disc	Original Size	7-Zip Size	Our Method Size	7-Zip *k*	Our Method *k*
D1	493,146,192	459,200,084	1,718	0.069	**0.999**
D2	752,425,968	580,962,585	2,612	0.228	**0.999**
D3	788,886,672	661,481,962	1,261,927	0.161	**0.998**
D4	830,822,832	828,265,184	1,297,480	0.003	**0.998**
D5	434,924,784	393,470,350	1,509	0.095	**0.999**
D6	724,778,208	647,053,819	2,524	0.107	**0.999**
D7	758,661,120	732,741,161	2,636	0.034	**0.999**
D8	664,190,688	505,434,682	2,313	0.239	**0.999**
D9	730,688,784	719,881,746	2,547	0.015	**0.999**
D10	347,736,144	340,546,094	1,291,617	0.021	**0.996**

to approximately reconstruct the intermediate scrambled data from the final dump using the standard scrambling table. To that end, our method first builds an approximate reconstruction of the scrambled intermediate data from the final dump, and our method then encodes the differences between the approximate reconstruction and the actual intermediate data.

Our method achieved a substantial space savings increase compared to compressing the intermediate data using 7-Zip's Ultra compression, with our method achieving a higher level of space savings for every disc tested. Thus, we believe our method will prove useful for easing the data storage burden encountered by those preserving CD-ROMs.

ACKNOWLEDGMENTS

We wish to thank the members of the data preservation communities.

REFERENCES

[1] M. Guttenbrunner, C. Becker, and A. Rauber, "Keeping the game alive: Evaluating strategies for the preservation of console video games," *International Journal of Digital Curation*, vol. 5, no. 1, Jun. 2010. [Online]. Available: https://doi.org/10.2218/ijdc.v5i1.144

[2] F. Cifaldi, ""It's just emulation!" - The challenge of selling old games," in *Game Developers Conference*, 2016. [Online]. Available: https://www.gdcvault.com/play/1023470/contactUs

[3] J. Newman, "The music of microswitches: Preserving videogame sound—a proposal," *The Computer Games Journal*, vol. 7, no. 4, pp. 261–278, 2018. [Online]. Available: https://doi.org/10.1007/s40869-018-0065-8

[4] N. Nylund, P. Prax, and O. Sotamaa, "Rethinking game heritage–towards reflexivity in game preservation," *International Journal of Heritage Studies*, vol. 27, no. 3, pp. 268–280, 2021. [Online]. Available: https://doi.org/10.1080/13527258.2020.1752772

[5] E. P. Conaway, *Server Worlds: Preservation, Virtualization, and Infrastructures of Control in Online Gaming*. University of California, Irvine, 2021.

[6] "Data interchange on read-only 120 mm optical data disks (CD-ROM)," Ecma International, Geneva, Switzerland, Standard, Jun. 1996. [Online]. Available: https://www.ecma-international.org/publications-and-standards/standards/ecma-130/

[7] L. B. Vries and K. Odaka, "CIRC-the error-correcting code for the compact disc digital audio system," in *Audio Engineering Society Conference: 1st International Conference: Digital Audio*. Audio Engineering Society, 1982.

[8] K. Immink, "Modulation systems for digital audio discs with optical readout," in *ICASSP '81. IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 6, 1981, pp. 587–589.

[9] "Multimedia command set - 5 (MMC-5)," T10, Washington, D.C., USA, Standard, Oct. 2006. [Online]. Available: http://www.t10.org/cgi-bin/ac.pl?t=f&f=mmc5r04.pdf

[10] Redump.org Community, "Combined offset in eac," 2010, accessed Jun. 15, 2022. [Online]. Available: http://forum.redump.org/topic/7649/combined-offset-in-eac/

[11] Accuraterip.com, "Accuraterip," 2010, accessed Jun. 15, 2022. [Online]. Available: http://www.accuraterip.com/

[12] Redump.org Community, "CD dumping guide with audio tracks (old)," 2022, accessed Jun. 15, 2022. [Online]. Available: http://wiki.redump.org/index.php?title=CD_Dumping_Guide_with_Audio_Tracks_(Old)&oldid=46420

[13] ——, "DiscImageCreator: Optical disc drive compatibility," 2022, accessed Jun. 15, 2022. [Online]. Available: http://wiki.redump.org/index.php?title=DiscImageCreator:_Optical_Disc_Drive_Compatibility&oldid=48878

[14] sarami, "DiscImageCreator," https://github.com/saramibreak/DiscImageCreator, 2022, accessed Jun. 15, 2022.

[15] K. Kaspersky, *CD Cracking Uncovered: Protection Against Unsanctioned CD Copying*. Wayne, PA, USA: A-List Publishing, 2004.

[16] Human Head Studios, "Rune," 2000, accessed Jun. 15, 2022. [Online]. Available: https://web.archive.org/web/20130622083143/http://www.rune-world.com/

[17] Redump.org Community, "Moderating guidelines for IBM PC and other systems," 2021, accessed Jun. 15, 2022. [Online]. Available: http://wiki.redump.org/index.php?title=Moderating_guidelines_for_IBM_PC_and_other_systems&oldid=45839

[18] ——, "Issues dumping pc disc with "code lock" copy protection," 2021, accessed Jun. 15, 2022. [Online]. Available: http://forum.redump.org/topic/29842/issues-dumping-pc-disc-with-code-lock-copy-protection/page/2/

[19] J. P. MacDonald, "xdelta: open-source binary diff, differential compression tools, VCDIFF (RFC 3284) delta compression," 2016, accessed Jun. 15, 2022. [Online]. Available: http://xdelta.org/

[20] I. Pavlov, "7-Zip," 2021, accessed Jun. 15, 2022. [Online]. Available: https://www.7-zip.org/