CONTENT-DEPENDENT CHUNKING FOR DIFFERENTIAL COMPRESSION, THE LOCAL MAXIMUM APPROACH

NIKOLAJ BJØRNER, ANDREAS BLASS, AND YURI GUREVICH

Abstract. When a file is to be transmitted from a sender to a recipient and when the latter already has a file somewhat similar to it, remote differential compression seeks to determine the similarities interactively so as to transmit only the part of the new file not already in the recipient's old file. Content-dependent chunking means that the sender and recipient chop their files into chunks, with the cutpoints determined by some internal features of the files, so that when segments of the two files agree (possibly in different locations within the files) the cutpoints in such segments tend to be in corresponding locations, and so the chunks agree. By exchanging hash values of the chunks, the sender and recipient can determine which chunks of the new file are absent from the old one and thus need to be transmitted.

We propose two new algorithms for content-dependent chunking, and we compare their behavior, on random files, with each other and with previously used algorithms. One of our algorithms, the local maximum chunking method, has been implemented and found to work better in practice than previously used algorithms.

Theoretical comparisons between the various algorithms can be based on several criteria, most of which seek to formalize the idea that chunks should be neither too small (so that hashing and sending hash values become inefficient) nor too large (so that agreements of entire chunks become unlikely). We propose a new criterion, called the slack of a chunking method, which seeks to measure how much of an interval of agreement between two files is wasted because it lies in chunks that don't agree.

Finally, we show how to efficiently find the cutpoints for local maximum chunking.

1. Introduction

The proliferation of networks such as intranets, extranets, and the internet has led to a large growth in the number of users that share information across wide networks. However, the amount of data that is transferred over the networks is still limited by cost and bandwidth constraints. As a result of limited network bandwidth, users can experience long delays or high costs in retrieving and transferring large amounts of data across a network.

Fortunately, there are stratagems for reducing the amount of data that must be transmitted. Data compression algorithms take advantage of the redundancy that is present in many files. They allow one to transmit not the file itself but information enabling the recipient to reconstruct the file; because of redundancy, this information may be much shorter than the original file.

In this paper, we are concerned with taking advantage of another frequently occurring situation, namely that the recipient already has a file similar to the one being transmitted. The idea here is, of course, to transmit only the new content, not the content that the recipient already has. Because what is transmitted is just the part of one file that differs from the other, one calls this compression method "differential compression". What makes differential compression non-trivial is that, to implement this idea, one must first decide what part of the file doesn't need to be sent, and one must decide this without sending massive amounts of information.

The most favorable situation of this sort arises, for example, in the distribution of software updates. Here, a client computer can tell a distribution server "Please update my program X to version 3.14159[1]; I currently have version 2.71828." The distribution server has a copy of the old version 2.71828, compares it with the new version 3.14159, and sends the client essentially just the difference between the two [27, 2, 30, 15, 16]. What makes this situation so pleasant is that the sender knows exactly what file the recipient already has (and that the sender got this information via a very short message from the recipient). "Local differential compression" refers

[1]This choice of version number is used by TEX.

to this situation where, after the initial message from the recipient, the sender can locally decide what information needs to be sent. Copious use of local differential compression is found in source and revision control systems [25, 29], in file systems [19], and in domain specific versions, such as binary code differencing used for distributing software patches [9, 21].

We shall treat the more difficult situation where the recipient has a file F_1 that is believed to be similar to the file F_2 to be transmitted, but the exact content of F_1 is not known to the sender. We shall discuss some existing protocols and propose new ones for handling this situation, i.e., for taking advantage of similarity between F_1 and F_2 to reduce the amount of data being sent. "Remote differential compression" (RDC) refers to such protocols, where the difference between the two files cannot be produced by the sender alone but must be determined interactively by the sender and the recipient. We also suggest a new measure, which we call slack, for comparing such protocols. And we compare the various protocols using several criteria, including slack.

A comprehensive overview of how chunking is used for RDC, how RDC can be used recursively, how similar files are identified for RDC, and how RDC can be used within a file replication system is presented in [28].

All the RDC protocols that we consider involve dividing at least one of the files into segments, which we call *chunks*, and computing hash values of these chunks to determine which chunks are common between the sender's and the recipient's files and thus do not need to be sent. The protocols differ in how the chunks are chosen.

Remark 1. RDC chunking protocols are designed for situations where reasonably large segments of F_1 and F_2 agree (but are perhaps in different locations in the two files). Other sorts of similarity would require other sorts of RDC protocols.

One sort of similarity that may fail to produce agreement of significant chunks occurs in compilation of programs. Compilers produce binaries with jump statements, where the jump locations are offsets into the file. These offsets are represented as absolute numbers. As a result, two almost equal source programs may compile to binaries with differences rather densely distributed throughout the files.

Something similar happens when files are compressed. Local differences between two files may result in densely distributed differences between their compressed versions.

Another sort of example arises from different methods of encoding. If a single file is encoded according to two different schemes, then the two encoded files are certainly similar in an intuitive sense, but that similarity may not result in any actual agreement between the encoded files.

All these situations are outside the scope of this paper. We are concerned here only with the use of chunking to detect and exploit agreement between reasonably long segments of two files.

The RSYNC protocol [31] uses the simplest and uniform choice of chunks: the recipient chops his file into chunks that are all of the same length l (except for the last chunk if l doesn't divide the file length). He then sends a weak and a strong, collision resistant checksum (or hash value — we use the terms interchangeably) of each segment to the sender. The sender traverses his version of the file, computing weak checksums over a sliding window. The weak checksums are used in a crude, in-cache filter to find candidates to match the chunks hashed by the recipient. By using the strong checksums to validate the candidate local file chunks, the sender can then deduce which chunks the recipient already has and which parts of the file need to be transferred directly.

Note that, in this approach, it is necessary for the sender to compute weak checksums for all segments of length l in his file. It would not do for the sender to chop his file into chunks of length l as the recipient did and to compute checksums only for those chunks. Two files that differ merely by adding a single character at the beginning would almost surely have no chunks in common, so the nearly total similarity of the files would be entirely wasted.

The sender thus has to do considerable work, computing the hashes of all length l segments of the file he wants to send, and comparing the results with the list of hash values obtained from the recipient.

If a protocol of this sort is to be used repeatedly, to transfer F_2 to many recipients, who have different approximations F_1, then the sender must either repeat all this work for each recipient or else store all the hash-values (considerably more data than the original file F_2) and then still do the comparisons separately for each recipient.

Remark 2. One could try to alleviate these problems by (partially) reversing roles. Let the sender chop F_2 into chunks of fixed size and send weak and strong hashes of these to the recipient. The recipient computes weak hashes in a sliding window to find chunks that might already be in F_1. After using the strong checksums to confirm the candidates, he asks the sender for those chunks that he doesn't already have. Of course, the total amount of work and communication here is essentially the same as in RSYNC, but if there are many recipients then much of the work is distributed among them, rather than being completely loaded on the sender. On the other hand, in many applications, such as updating calendar schedules or mailboxes, the file transfer is, from the recipients' point of view, mere overhead, not part of their immediate work. So it may be inappropriate to assign most of the work to the recipients.

Remark 3. It is possible to reduce the communication overhead of the RSYNC protocol by using multiple rounds [18]. In the first round, use a relatively large chunk size. If there are large segments that match, they will be handled during this round. Subsequent rounds use progressively smaller chunk sizes.

Remark 4. We are concerned in this paper with reducing the communication needed for file transfer, but in particular applications other considerations may become especially important. For example, when files are sent to space-constrained devices, standard RSYNC has the drawback of requiring the receiver to create a fresh copy of the received file. For this situation, an in-place version of RSYNC is proposed in [24] to reduce the recipient's storage needs.

The protocols that we treat in this paper, known ones as well as new, proceed differently from RSYNC in that both the sender and the recipient divide their files into chunks and compute (strong) checksums for the chunks. To avoid the pitfall described above, where a single character added to a file can make the chunks entirely different, the chunks in these protocols are not of a fixed length; rather, the places where the file is to be cut, the *chunk boundaries*, are determined by internal features of the files. This is the meaning of "content-dependent chunking".

The protocols under consideration all proceed according to the following rough outline; details will be added later. As before, we use F_2 to denote the file to be transmitted and F_1 to denote a file that is already owned by the intended recipient of F_2 and that is believed to have substantial overlap with F_2.

(1) The sender chops F_2 into chunks and computes a hash value for each chunk.
(2) The recipient does the same for F_1.
(3) The sender sends the recipient the hash values for F_2 (along with the lengths of the chunks).
(4) The recipient compares those hash values with the ones he computed for F_1. When two agree, he assumes that the corresponding chunks of F_1 and F_2 are the same, so there is no need for the sender to transmit those chunks of F_2.
(5) The recipient tells the sender which of the chunks of F_2 need to be sent.
(6) The sender sends those chunks.

Remark 5. We describe in this remark a situation where content-dependent chunking has an important advantage over protocols like RSYNC that require a sliding window rather than independent chunking by the two parties. The situation is that the recipient is believed to have parts of F_2 in some file (or files) somewhere in his system, but it is not known where. In other words, the recipient doesn't know which file should be F_1 (or perhaps he should use several files, each containing some part(s) of F_2). With a content-dependent protocol, the recipient can prepare (ahead of time) a list of hashes of all the chunks of all his relevant files. When he gets from the sender the hashes of the chunks of F_2, he compares these with the contents of his list. If one tried to apply this idea to RSYNC, the recipient would send that whole list to the sender, who would have to compare everything listed with all the hashes produced in his sliding window. For the role-reversed variant of RSYNC, the situation is even worse; if the recipient wanted to prepare a list in advance, it would have to contain all the (weak) checksums of all the contents of the sliding window in all the relevant files.

Remark 6. The communication cost of content-dependent chunking can be reduced by using a chunking method recursively as follows [28]. Fix the parameter(s) of the chunking method to

yield a relatively small chunk size c, just large enough to make it worthwhile to compute and send hashes of such chunks rather than the chunks themselves. Let Chk denote the length of a hash value. Apply the chunking method to produce a sequence of checksums, whose concatenation we regard as a new file $F^{(1)}$. On average, c symbols in F are represented by Chk symbols in $F^{(1)}$. Now apply the chunking protocol to $F^{(1)}$, obtaining a new file $F^{(2)}$. On average, Chk symbols here represent c symbols in $F^{(1)}$, i.e., c/Chk hash values in $F^{(1)}$, and thus c^2/Chk symbols in the original F. Repeating the process n times, we get a file $F^{(n)}$, each Chk symbols of which represent, on average c^n/Chk^{n-1} symbols of F. By choosing the number n of iterations suitably, we can arrange that each checksum in $F^{(n)}$ represents a rather large chunk of F. Now, to transmit F, the sender should first send $F^{(n)}$. When a hash value here matches one in the file already owned by the recipient, a large chunk of F has been transferred. For those hash values in $F^{(n)}$ that don't match any of the recipient's, the sender should next transfer the chunks of $F^{(n-1)}$ that were hashed to produce those values. Continue similarly for n rounds, sending the necessary chunks from $F^{(k)}$ for smaller and smaller k, where "necessary chunks" are those whose hashes didn't match any of the recipient's at the previous round. At the very end, when k has decreased to zero, send the remaining chunks of the original file F.

This sort of repeated chunking and hashing, converting F into $F^{(1)}$, then into $F^{(2)}$, and so on, would not work with a fixed chunk length protocol such as RSYNC. As we saw earlier, addition of a single character at the start of a file would completely change the checksums in $F^{(1)}$. As a result, all the later files $F^{(k)}$ would also be completely different, and the entire similarity between the files would be wasted. When one uses RSYNC repeatedly, sending large chunks first and then smaller ones as in Remark 3, it is necessary to process the whole original file F for each of the desired chunk sizes. With content-dependent methods, the files to be processed decrease in length at each step, by a factor c/Chk.

The main difference between the various protocols we consider will be the *chunking methods*, i.e, how the files are to be divided into chunks at steps (1) and (2) in the outline above. In step (3), the sender should provide the lengths of the chunks of F_2 because these will not be fixed by the chunking method. He should also provide the locations in F_2 of those chunks (their *offsets*) if these cannot easily be computed from the lengths (e.g., if the information about different chunks might be received out of order). Then in step (5), the recipient can efficiently request the necessary chunks by sending their offsets to the sender.

This somewhat rough description of content-dependent chunking protocols makes some desiderata evident. First, the chunks should not be too short. The main reason is that the efficiency of the protocol depends on sending hash values that are significantly shorter than the chunks they represent. The hash values cannot be too short, lest accidental coincidences of hash values lead the recipient to think he already has a chunk when he doesn't. And the chunks themselves should be a good deal longer than the hash values; otherwise one might as well send the chunks themselves (i.e., send all of F_2) rather than computing and sending hash values.

There are other disadvantages associated with short chunks. One is that strong checksums have to be re-initialized for each small chunk, so setting up the computation for each strong checksum has an overhead. More importantly, each checksum is stored in a table and the table is searched for matches with checksums from the other files. There is a time overhead in storing and searching checksums.

On the other hand, the chunks should not be too long. With excessively long chunks, there is a risk that large segments of F_1 and F_2 might coincide yet no whole chunk coincides. Then the protocol would not detect any of the agreement between the two files, and the recipient would have to request all the chunks (i.e., all of F_2) from the sender.

A third desideratum is that similar files should be chopped into chunks at corresponding locations. Similarity of the files does us no good if the protocol fails to detect the similarity because the files were chopped into entirely different chunks. It is this requirement that prevents us from using chunks all of the same length in both files.

We shall describe and analyze a standard content-dependent chunking method, the one used in the Low Bandwidth File System (LBFS) proposed in [20], and we shall propose and analyze two new content-dependent chunking methods, called interval filter chunking and local maximum chunking. The analyses of these methods involve several measures, related to the desiderata

described above. For example, since excessively long chunks and excessively short chunks both cause problems, it is desirable to keep the variance of the chunk length (on random files) low. For similar reasons, one may want to reduce the probability of getting chunks a great deal longer than the average chunk. We also introduce a more precise measure, though unfortunately rather difficult to compute, the slack of a chunking method, which takes into account not only the lengths of chunks but also the method's ability to take advantage of identical segments in files by putting chunk boundaries in matching places.

Let us say a few words on the history of the RDC project. It was conceived by and executed during 2003–2005 in the Core File Systems group in the Windows division of Microsoft. The project was successful, and the technology is widely used in Microsoft products. Results were reported in technical report [28]. The local-max method was the result of a collaboration of the Core File Systems group and Microsoft Research. The group performed experiments in order to evaluate different chunking methods. The local-max method proved to be superior to the competitors. The question arose whether there were a priori, mathematical reasons behind the better performance of the local-max method. We conceived this paper as a mathematical account with a relatively narrow purpose to clarify various issues related to content-dependent chunking. The narrow purpose is reflected in the title of the paper; the paper is not about the RDC project in general.

In more detail, the content of this paper is as follows. In Section 2, we present some preliminary information, including some mathematical tools needed later and some conventions concerning the files we consider. In Section 3, we introduce a simple probabilistic model of files with partial agreement, and we use it to define a measure, which we call slack, of the responsiveness of a chunking method to agreements between the files. That is, once two files start to agree, how much further in the files must one go until whole chunks agree? Section 4 is devoted to a description and analysis of point-filter methods, particularly the method used in LBFS [20]. In Section 5, we introduce and study one of our proposed new chunking methods, the interval filter method. Section 6 does the same for our second (and better) new method, the local maximum method. Section 7 is about the probabilities, under various chunking methods, of finding long intervals without any chunk boundaries. In Section 8, we give an efficient algorithm for finding the chunk boundaries in the local maximum method. (For the other methods, efficient algorithms are easy to see, but for the local maximum method this matter is not trivial.) Section 9 describes an experimental evaluation of the chunking methods in the context of Microsoft's RDC protocol. We also report a few experiments for evaluating chunking methods in isolation. Finally, in Section 10, we indicate connections with other work.

Applicability. Local maximum chunking is used as part of the RDC algorithms included in the Distributed File System Replication engine that was released as part of Windows Server 2003 R2 [4]. It is also being used as part of the file replication engine underlying Sharing Folders in the Windows Live Messenger 8.0 [5], and as part of Windows Meeting Space in Windows Vista [7]. The RDC algorithms are furthermore packaged as a stand-alone library that is made publicly available for application developers [6].

2. Preliminaries

We collect in this section our conventions about files in general and random files in particular. We also recall some facts from probability theory, including ergodic theory, some formulas that will be used in our calculations, and some combinatorial information about greedy sequences. The reader may refer to [11], [22], and [12] for further information about these topics.

2.1. **Files.**

In the description and analysis of content-dependent chunking protocols, we shall use the following model of files.

We model a file as a sequence of elements from a finite set PFE of *potential file entries*. In reality, the sequence is always finite, its positions being indexed by a segment $[0, l-1]$ of the natural numbers. (It is convenient to start the indexing at 0 rather than 1; we stop at $l-1$ so that l denotes the length.) We shall, however, sometimes use infinite sequences, indexed by the set $\mathbb{N}$ of all natural numbers, or even doubly infinite sequences, indexed by the set $\mathbb{Z}$ of all integers.

Infinite and doubly infinite sequences serve as a convenient mathematical approximation to long finite sequences.

When we use the words "left" and "right", in connection with the positions in a file, we always assume the traditional picture of $\mathbb{Z}$; the integers lie on a horizontal line, with the smaller ones to the left of the larger ones. For example, we would call 0 the left end and $l-1$ the right end of the interval $[0, l-1]$.

For our analyses of various chunking methods, we shall assume that the entries of a file are probabilistically independent and uniformly distributed. That is, if I denotes the index set (a segment $[0, l-1]$ or $\mathbb{N}$ or $\mathbb{Z}$), then we give the space PFE^I of files the product measure determined by the uniform measure on PFE. This means in particular that, if $i_1, \dots, i_k$ are distinct elements of I (i.e., distinct positions in a file), if $X_1, \dots, X_k$ are subsets of PFE, and if A is the set of those files F for which $F(i_j) \in X_j$ for each j (i.e., the entries at the positions i_j come from the corresponding sets X_j, all other entries being unconstrained), then A has probability (or measure)

$$\mathbf{Prob}(A) = \frac{\prod_j |X_j|}{|\mathrm{PFE}|^{|I|}},$$

where $|X|$ means the number of elements in the set X.

For finite I, it follows that any subset A of PFE^I has probability $|A|/(|\mathrm{PFE}|^{|I|})$; that is, we have the uniform distribution on files. For infinite I, the laws of probability theory provide a unique measure, not for all subsets of PFE^I but for all reasonably well-behaved ones (known as measurable sets or as events). The measurable sets include all the sets that will arise in our discussion. This measure is also called the uniform measure, just as for finite I, even though it cannot be defined by simply saying that all individual elements of PFE^I have the same probability. (They do have the same probability, but it is zero, and so it tells us nothing about probabilities of more interesting events.)

We use standard terminology and notation from probability theory. For example, when A is a measurable set, we say that a random file has probability $\mathbf{Prob}(A)$ of being in the set A. When this probability is 1, then we say that files are *almost surely* in A and that *almost all* files are in A. We use $\mathbf{E}(f)$ for the expectation and $\mathbf{Var}(f)$ for the variance of a random variable (i.e., a measurable, real-valued function on the probability space). We also use the standard notations $\mathbf{Prob}(A|B)$, $\mathbf{E}(f|B)$, and $\mathbf{Var}(f|B)$ for the conditional probability, expectation, and variance, conditional on the event B, assumed to have positive measure.

Whether the product measure accurately reflects the actual probabilities of files in the real world depends on the sort of files under consideration. Highly compressed files are close to random in our sense, but English text files are not, for two reasons. First, the probabilities of individual characters are not equal; the letter q occurs far less often than the uniform measure predicts, while the space occurs far more often. Second, the probabilities at different locations in the file are not independent; for example, the probability of the letter u is far higher immediately after q than elsewhere. Similarly, spreadsheets tend not to be random, as they often have considerable periodic content.

Fortunately, experimentation has shown that our protocols, particularly the local maximum chunking, work well even on common sorts of files, like English text, where our analysis becomes doubtful because our randomness assumptions fail.

Remark 7. There are rather easy ways of increasing the apparent randomness of a file. Given a file that is a sequence of symbols from an alphabet Σ (not our intended alphabet PFE), one can compute a hash value for each contiguous subsequence of some fixed window size w. The resulting sequence of hash values constitutes a new file, whose set PFE is the set of all possible hash values. Because of the hashing, this new file usually looks random even if the original file did not.

The time needed to compute hash values for all the windows of length w can be reduced by using a *rolling hash function.* This means that the hash value for each window except the first is computed from the hash value h of the immediately preceding window, the first symbol a in that preceding window (the symbol that is no longer present in the new window), and the last symbol b in the new window (the symbol that was not in the previous window).

If we assume that the symbols in Σ can be represented by bit-vectors of length w, then we can obtain a very simple rolling hash, using bit vectors of length w as hash values, and using the operations of bitwise exclusive or and rotation on these vectors, as follows. Given the hash value h for a particular window, given the first symbol a in that window (which is about to leave the window), and given the next symbol b after that window (which is about to enter the window), regard a and b as bit vectors of length w. Compute the hash value of the next window by first taking the bit-wise exclusive-or $h \oplus a \oplus b$ and then rotating the resulting bit vector by one position (the last bit is removed and put in the front). Because the window size w equals the length of the bit vectors, when the element b that has just entered the window leaves the window w steps later, the hash vector will have been rotated by one full rotation. So the exclusive-or addition of b when it entered the window will be exactly canceled by the addition of b when it leaves the window.

A prime example of a rolling hash, for which the collision probabilities have been thoroughly analyzed, is the Rabin hash [23, 17]. It is based on arithmetic modulo an irreducible polynomial with coefficients in $\mathbb{Z}/2$. The number $|\mathrm{PFE}|$ of possible hash values can be adjusted by using polynomials of degree $\log(|\mathrm{PFE}|)$.

The local maximum chunking method was originally proposed and implemented with a preliminary rolling hash, intended to introduce the randomness that our analysis presupposes. (Strictly speaking, a deterministic, length-preserving transformation cannot introduce or increase randomness. It can, however, mask any regularities so that they are unlikely to influence the analysis of chunking protocols.) Later, it was found experimentally that the local maximum method works well even without this preliminary hashing.

Rolling hashes essentially summarize the contents of a neighborhood in each position of the file, thus making the new file more resistant to local entropy variations.

Remark 8. There are additional actions that one can undertake in order to increase the entropy. For example, Mark Manasse noticed that if a short pattern repeats many times in succession, as in a long stretch of zeros, then that stretch can be compressed to a much shorter string before rolling hashes are applied. The idea is to replace many successive occurrences of the same string with one occurrence and the number of times to repeat it. (Care is needed to avoid possible ambiguity of such repetition instructions, but we need not concern ourselves with the details here.) Such *run-length encoding* is essential for the content-dependent chunking methods discussed in this paper when the file exhibits periodicity with period significantly shorter than the horizon of the chunking method. If no coding is undertaken, then such periodicity would produce undesirably long chunks under the interval filter and local maximum methods, because there would be no cutpoint in the periodic stretch of the file. Under the LBFS method, there would be cutpoints, but identical periodic segments in two files might well have their cutpoints in entirely different places.

We make an additional assumption about our files, namely that the set PFE of potential file entries is equipped with a linear ordering. In many situations, this assumption is clearly satisfied. If the potential file entries are hash values, or integers obtained in some other way, then we can use the usual ordering of integers. If they are characters, then we can order them by their ASCII codes or Unicodes.

One might even argue that, in real computers, potential file entries are always linearly ordered because they are ultimately represented by bit strings, and we can use the lexicographic ordering of these strings. This observation works as long as the sender and recipient use the same bit string representations. We need our linear orderings to be the same for the sender and the recipient, and whether the computers' internal bit strings can serve this purpose depends on the particular application.

Having assumed a linear ordering of PFE, we obtain a canonical bijection between PFE and $\{0, 1, \dots, |\mathrm{PFE}| - 1\}$, namely the unique order-preserving bijection. We shall therefore, whenever it is convenient, assume without loss of generality that $\mathrm{PFE} = \{0, 1, \dots, |\mathrm{PFE}| - 1\}$.

2.2. **Ergodic Theory.** We shall use a little ergodic theory in part of our analysis, so we summarize here what is needed. We state the results in their natural generality, namely a probability

space Ω with a measure-preserving, one-to-one transformation T of Ω onto itself. In our applications of these results, however, Ω will always be the space $\mathrm{PFE}^{\mathbb{Z}}$ of doubly infinite files, and T will always be the (leftward) *Bernoulli shift*, BS, which sends any file $F \in \mathrm{PFE}^{\mathbb{Z}}$ to the file $G = \mathrm{BS}(F)$ defined by $G(i) = F(i+1)$. (The reader should see that, despite the impression one might get from the plus sign in $i+1$, this really does shift a file to the left.) Thus, the reader can safely pretend that whenever we write Ω and T, we mean $\mathrm{PFE}^{\mathbb{Z}}$ and BS. Clearly, the Bernoulli shift is a one-to-one function from $\mathrm{PFE}^{\mathbb{Z}}$ onto itself. (Indeed, this is a major reason for using doubly infinite files.) It is also clear, from the definition of the probability measure on $\mathrm{PFE}^{\mathbb{Z}}$, that this measure is invariant under BS. (So BS is an automorphism of the probability space.)

A measure-preserving bijection $T : \Omega \to \Omega$ is called *ergodic* if, whenever an event $A \subseteq \Omega$ is invariant (meaning $T(A) = A$), then its probability is 0 or 1. It is known that BS is ergodic (see [22, Section 2.4, Example (1)]), so all the following results about ergodic transformations apply to the particular case that we need later. Notice that the definition of ergodicity would be unchanged if we required probability 0 or 1 for all events A for which $T(A) \subseteq A$. This is because T is measure-preserving, so such an A would differ from $T(A)$ by a set of measure 0, and the intersection $\bigcap_{n\in\mathbb{N}} T^n(A)$ would be an invariant set differing from A by a set of measure 0.

We shall need three classical theorems of ergodic theory. Poincaré's Recurrence Theorem [22, Theorem II.3.2] implies that, if $T : \Omega \to \Omega$ is ergodic and if $A \subseteq \Omega$ is an event of positive probability, then almost all points $x \in \Omega$ have the property that $T^k(x) \in A$ for some positive integer k (in fact for infinitely many k). Birkhoff's ergodic theorem [22, Theorems II.2.3 and II.4.4] gives more detail about how often the sequence $T^k(x)$ visits A.

Proposition 9. *Let $T : \Omega \to \Omega$ be ergodic and let $A \subseteq \Omega$ be any event. Then for almost all $x \in \Omega$,*

$$\lim_{N\to\infty} \frac{\text{Number of } k \in [0, N-1] \text{ with } T^k(x) \in A}{N} = \mathbf{Prob}(A).$$

Another way to measure frequency of visits to A is the time until the first visit to A. For ergodic T, let $\rho(x)$ denote the least $k \geq 1$ with $T^k(x) \in A$. (Either define $\rho(x)$ arbitrarily on the measure-zero set of points x for which no such k exists, or simply ignore sets of measure zero.) Kac's theorem [22, Theorem II.4.6] gives the following result. Note that it is about random elements of A, not of the whole space Ω; that is, the expectation in the conclusion of the theorem is conditional on $x \in A$.

Proposition 10. *Let A be an event of positive probability p in Ω, and let x be a random member of A. The expectation of the return time $\rho(x)$, $\mathbf{E}(\rho|A)$, equals $1/p$.*

It will be useful to have a companion result to Kac's theorem, giving the expectation of ρ on the whole space Ω rather than on A. Easy examples show that this $\mathbf{E}(\rho)$ is not determined by $\mathbf{Prob}(A)$ alone, but it turns out to be related to the variance of ρ on A.

Proposition 11. *With notation as above,*

$$\mathbf{E}(\rho) = \frac{1}{2}\left[\mathbf{Prob}(A)\,\mathbf{Var}(\rho|A) + \frac{1}{\mathbf{Prob}(A)} + 1\right]$$

Proof. For the sake of brevity, we systematically ignore sets of measure zero; they do not affect any of the following computations. Partition A into the pieces $A_n = \{x \in A : \rho(x) = n\}$ $(n \in \mathbb{N} - \{0\})$.

It is not difficult to see that the sets $T^k(A_n)$ for $0 \leq k < n$ are all pairwise disjoint. Indeed, suppose, toward a contradiction, that we had $x \in T^k(A_n) \cap T^{k'}(A_{n'})$, and suppose this counterexample is chosen with k as small as possible. If neither k nor k' were 0, then $T^{-1}(x) \in T^{k-1}(A_n) \cap T^{k'-1}(A_{n'})$ would contradict the minimality of k. So we may assume $k = 0$ and so $x \in A_n \cap T^{k'}(A_{n'})$. In particular, $x \in A$ and $x = T^{k'}(y)$ for some $y \in A_{n'}$. But then from $k' < n' = \rho(y)$ we get $T^{k'}(y) \notin A$, a contradiction.

Consider how T acts on the sets $T^k(A_n)$ for $0 \leq k < n$. It sends each one to the one with k increased by 1, except that when $k = n-1$ it sends $T^{n-1}(A_n)$ into A, according to the definitions of A_n and ρ. Since A is the union of the various A_n's, it follows that $\bigcup_{0\leq k<n} T^k(A_n)$ (where both n and k vary) is mapped into itself by T. By ergodicity, its measure is 0 or 1. As it includes A,

its measure cannot be 0, so it is almost all of the space Ω. Since we are ignoring sets of measure 0, we can say that Ω is partitioned into the sets $T^k(A_n)$, where, as before, $0 \le k < n$.

From the definitions of A_n and ρ, it follows that ρ is constant on $T^k(A_n)$ with value $n-k$. Therefore,

$$\mathbf{E}(\rho) = \sum_{0\le k<n} (n-k)\,\mathbf{Prob}(T^k(A_n)) = \sum_{0\le k<n} (n-k)\,\mathbf{Prob}(A_n),$$

where the second equality uses the fact that T preserves the measure. Carrying out the summation over k for each fixed n, we get

$$\mathbf{E}(\rho) = \sum_{n=1}^{\infty}\left(\mathbf{Prob}(A_n)\frac{n(n+1)}{2}\right) =$$

$$= \frac{1}{2}\left[\sum_n n\,\mathbf{Prob}(A_n) + \sum_n n^2\,\mathbf{Prob}(A_n)\right].$$

The first of the two sums in the brackets here can be rewritten as the sum of $\mathbf{Prob}(T^k(A_n))$ over all n and all $k < n$. So, as these sets $T^k(A_n)$ partition Ω, this sum is simply 1. (This observation is essentially a proof of Kac's theorem.) The second sum can be rewritten as follows, using the fact that, since $A_n \subseteq A$, $\mathbf{Prob}(A_n) = \mathbf{Prob}(A)\,\mathbf{Prob}(A_n|A)$.

$$\mathbf{Prob}(A)\sum_n n^2\,\mathbf{Prob}(A_n|A) = \mathbf{Prob}(A)\,\mathbf{E}(\rho^2|A).$$

Furthermore, since all random variables satisfy $\mathbf{Var}(f) = \mathbf{E}(f^2) - \mathbf{E}(f)^2$, we can rewrite this in terms of the variance as

$$\mathbf{Prob}(A)[\mathbf{Var}(\rho|A) + \mathbf{E}(\rho|A)^2].$$

Remembering that $\mathbf{E}(\rho|A) = 1/\,\mathbf{Prob}(A)$ by Kac's theorem, and substituting the results of our computation back into the formula for $\mathbf{E}(\rho)$, we immediately get the proposition. $\square$

2.3. **Useful Formulas.** We collect here some formulas for use in the calculations in later sections. First, there is the well-known formula for the sum of a geometric series:

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x} \qquad \text{for } |x| < 1.$$

Differentiating term by term (which is correct in the interior of the interval of convergence of any power series) and omitting the vanishing $i = 0$ term from the result, we get

$$\sum_{i=1}^{\infty} i x^{i-1} = \frac{1}{(1-x)^2} \qquad \text{for } |x| < 1.$$

Multiplying this equation by x and then differentiating again, we get

$$\sum_{i=1}^{\infty} i^2 x^{i-1} = \frac{1+x}{(1-x)^3} \qquad \text{for } |x| < 1.$$

We also need a special case of the familiar formula for the sum of an arithmetic progression.

$$\sum_{m=1}^{h} m = \frac{h(h+1)}{2}.$$

A similar formula for adding values of a quadratic polynomial will be useful in the form

$$\sum_{m=2}^{h} m(m-1) = \frac{h^3-h}{3}.$$

Although this formula, once it is proposed, is easily proved by induction, it can also be seen directly, in the equivalent form

$$\sum_{m=2}^{h}\binom{m}{2} = \binom{h+1}{3}.$$

The right side here counts the 3-element subsets of $\{0, 1, \ldots, h\}$. The term with index m on the left side counts those 3-element subsets whose last element is m, since such a subset is determined by its other elements, a 2-element subset of $\{0, 1, \ldots, m-1\}$. (The same proof gives the corresponding result for binomial coefficient sums with 2 and 3 changed to any k and $k+1$.) Note that the lower limit $m = 2$ in the sums here can be changed to 1 or to 0, since the corresponding terms $m(m-1)$ vanish.

We shall also have use for estimates of sums of powers in the form

$$\sum_{k=0}^{m-1} \frac{1}{m}\left(\frac{k}{m}\right)^r.$$

This is a lower Riemann sum approximating the integral

$$\int_0^1 x^r\,dx = \frac{1}{r+1}.$$

An upper Riemann sum approximating the same integral is obtained by letting k range from 1 to m rather than from 0 to $m-1$. That amounts to adding $1/m$ to the sum, so we have

$$\frac{1}{r+1} - \frac{1}{m} < \sum_{k=0}^{m-1} \frac{1}{m}\left(\frac{k}{m}\right)^r < \frac{1}{r+1}.$$

So for large m we have

$$\sum_{k=0}^{m-1} \frac{1}{m}\left(\frac{k}{m}\right)^r \approx \frac{1}{r+1}.$$

Although these approximations suffice for our needs, we note that there is an exact expression in terms of the Bernoulli numbers B_k:

$$\begin{aligned}\sum_{k=0}^{m-1} \frac{1}{m}\left(\frac{k}{m}\right)^r &= \frac{1}{m^{r+1}}\frac{1}{r+1}\sum_{k=0}^{r}\binom{r+1}{k}B_k m^{r+1-k}\\ &= \frac{1}{r+1} - \frac{1}{2m} + \frac{r}{12m^2} + \ldots.\end{aligned}$$

The first term here is the integral approximation obtained above, and the second term says that the integral is approximately halfway between the upper and lower Riemann sums.

2.4. **Greedy Increasing Sequences.** Consider a finite file F or, more generally, a function F from any interval of integers $I = [a, b]$ to PFE. Recall that we identified the set PFE of potential file entries with a set of integers $\{0, 1, \ldots, |\mathrm{PFE}| - 1\}$. So it makes sense to talk about increasing (or decreasing) subsequences of F. It will be convenient here to discuss subsequences, not in terms of the values of F that constitute them, but in terms of the positions where those values occur.

For a fixed F, we define the *left-to-right greedy increasing sequence*, abbreviated *→greedy sequence*, in the interval $I = [a, b]$ as follows. Its first element g_0 is the first element a of I. Thereafter, g_{k+1} is defined as the smallest $n \in I$ such that $n > g_k$ and $F(n) > F(g_k)$. That is, we build an increasing sequence of elements of I such that the corresponding sequence of F-values is also increasing, and we do so greedily, always putting into our sequence the first available number. The sequence ends at g_k when there is no n satisfying the requirements for g_{k+1}. Notice that then $F(g_k)$ is the largest value that F attains on I.

There is an analogous definition of the *right-to-left greedy sequence* or *←greedy sequence*, which starts with the rightmost point b of I and thereafter takes g_{k+1} to be the rightmost point n to the left of g_k with $F(n) > F(g_n)$. Notice that the terms in this sequence are chosen in decreasing order, but their F-values are in increasing order, so that the corresponding restriction of the original sequence is decreasing. Because of this somewhat confusing situation, we do not use the words "increasing" or "decreasing" in connection with ←greedy sequences.[2]

[2] There are a half dozen more variants of greediness: We could take, in either the left-to-right or the right-to-left versions, successive points with smaller rather than larger F-values. And in all these situations, we could use weak rather than strict inequalities of F-values. We shall get by with the →greedy and ←greedy sequences as defined above and avoid needing any of these other variants.

When we simply say "greedy sequence" without further modifiers, we mean the $\rightarrow$greedy increasing sequence.

The elements g_k of the greedy increasing sequence admit a simple alternative characterization that does not involve recursion on k.

Proposition 12. *The $\rightarrow$greedy increasing sequence in I consists of those $n \in I$ such that $F(n) > F(m)$ for all $m \in I$ such that $m < n$.*

In other words, they are the places where, as we read the sequence F from left to right, we see a new maximum value. We shall refer to such places as *left-to-right maxima* or $\rightarrow$*maxima* of F. Analogously, of course, the $\leftarrow$greedy sequence consists of the $\leftarrow$maxima, those $n \in I$ such that $F(n) > F(m)$ for all $m \in I$ such that $m > n$.

Proof. We first prove, by induction on k, that each g_k satisfies the condition in the lemma. For g_0, this is vacuously true, since there is no smaller $m \in I$. As for g_{k+1}, we have $F(g_{k+1}) > F(g_k)$ by definition and $F(g_k) > F(m)$ for all $m < g_k$ in I by induction hypothesis, so it remains only to consider $m \in (g_k, g_{k+1})$. But the greediness in the definition of g_{k+1} implies that all such m have $F(m) \leq F(g_k) < F(g_{k+1})$.

Conversely, suppose $n \in I$ satisfies the condition in the proposition, and let k be the largest index for which $g_k \leq n$. (This exists because g_0, being the first element of I, is $\leq n$.) If we had the strict inequality $g_k < n$, then, because g_{k+1} is chosen greedily and because $F(n) > F(g_k)$ by the assumption about n, we would have $g_{k+1} \leq n$, contrary to our choice of k. This contradiction shows that we must have $g_k = n$, and so the lemma is proved. □

Corollary 13. *If $n \in I$ is not a member of the $\rightarrow$greedy sequence, then there is some $m < n$ such that $F(m) \geq F(n)$ and m is a member of the $\rightarrow$greedy sequence.*

Proof. This was, in effect, proved in the second half of the proof of Proposition 12, but it can also be obtained by applying the proposition itself, as follows. If n is not in the greedy subsequence, then there is, by the proposition, some $m < n$ with $F(m) \geq F(n)$. The smallest such m is, by the proposition again, in the greedy sequence. □

We turn next to some probabilistic information about the $\rightarrow$greedy sequence for a random file $F : [a, b] \rightarrow \text{PFE}$. We shall use this information in a context where $|\text{PFE}| \gg b - a$ and therefore it is very unlikely that two positions in $[a, b]$ will have the same F-value. We take advantage of this circumstance by doing our calculations under the assumption that F is one-to-one, i.e., that there are no "ties" between F-values. Formally, this means that we work not with the usual probability space $\text{PFE}^{[a,b]}$ but with the subspace consisting of the one-to-one functions (and with the probability measure restricted to this subspace and renormalized to have total probability 1).

Proposition 14. *For any $n \in [a, b]$, the probability that n is in the $\rightarrow$greedy sequence is $1/(n - a + 1)$.*

Proof. For any $n \in [a, b]$, the largest F-value at the $n - a + 1$ positions in $[a, n]$ has an equal chance of occurring at any of these positions. So the probability that the largest F-value on $[a, n]$ occurs at n is $1/(n - a + 1)$. By Proposition 12, this is also the probability that n is in the greedy sequence. □

Corollary 15. *The expected length of the greedy sequence in $[a, b]$ is*

$$\sum_{n=a}^{b} \frac{1}{n - a + 1} = \sum_{m=1}^{b-a+1} \frac{1}{m} \approx \ln(b - a + 1) = \ln |[a, b]|.$$

Proof. The expected size of any set is the sum, over all potential elements n, of the probability that n is in the set. So the first sum in the corollary follows immediately from the proposition. The logarithm is a well-known asymptotic (for large $b - a$) approximation to these harmonic sums. □

Remark 16. It is known that the expected length of the *longest* (in contrast to the greedy) increasing sequence is $O(\sqrt{|[a, b]|})$. So the greedy method of selecting an increasing subsequence usually falls far short of the maximum achievable length. In fact, a theorem of Erdős and Szekeres

([10, 1]) asserts that a sequence of integers of length $pq+1$, without repetitions, must have an increasing subsequence of length $p+1$ or a decreasing subsequence of length $q+1$. So a sequence of length l will have a monotone sequence of length at least $\sqrt{l}$. By symmetry, there will be an increasing subsequence of this length at least half the time, so the expectation of the maximum length of an increasing subsequence is at least $\sqrt{l}/2$.

Although the greedy increasing sequences do not usually come near the length given by the Erdős-Szekeres theorem, they do provide an elegant proof of that theorem, as follows. Given a sequence F of distinct integers, let G_1 be the greedy increasing subsequence. Delete G_1 from F, and let G_2 be the greedy increasing subsequence of what remains. Continue in the same manner, forming and removing greedy increasing subsequences $G_3, \dots,$ until nothing remains. If one of these sequences G_i has at least $p+1$ terms, then we have the desired conclusion, so assume that each G_i has at most p terms. So the first q of our greedy sequences have altogether at most pq elements, not enough to exhaust the given sequence of length $pq+1$. Pick any position that is not in any of $G_1, \dots, G_q$. Because it was not in G_q, Corollary 13 provides an earlier position that has a larger F value and is in G_q. This, in turn, was not in G_{q-1}, so Corollary 13 provides an even earlier position with an even larger F value in G_{q-1}. Continuing in this way, we get a decreasing subsequence of length $q+1$ in F.

We shall need one additional, perhaps surprising piece of information about the $\rightarrow$maxima of a random file, namely that different positions behave independently.

Proposition 17. *For each $n \in I$, let E_n be the event that n is a $\rightarrow$maximum of a random file F. Then these events are probabilistically independent.*

Proof. Let $n_1 < n_2 < \cdots < n_k$ be elements of I. We must show that

$$\mathbf{Prob}(\text{All } n_i \text{ are } \rightarrow\text{maxima}) = \prod_{i=1}^{k} \mathbf{Prob}(n_i \text{ is a } \rightarrow\text{maximum}),$$

and we shall do this by induction on k, the cases $k=0$ and $k=1$ being trivial. Suppose, therefore, that the result holds for $k-1$. Consider the conditional probability

$$\mathbf{Prob}(n_1, \dots, n_{k-1} \text{ are } \rightarrow\text{maxima} \,|\, n_k \text{ is a } \rightarrow\text{maximum}).$$

The event that all of $n_1, \dots, n_{k-1}$ are $\rightarrow$maxima depends only on the relative order of the values of F at points $n < n_k$. The conditioning event, that n_k is a $\rightarrow$maximum, means that all those values $F(n)$ are smaller than $F(n_k)$, but it says nothing about the order of those $F(n)$'s relative to each other. Thus, the conditional probability equals the absolute probability that $n_1, \dots, n_{k-1}$ are $\rightarrow$maxima, which is, by the induction hypothesis,

$$\prod_{i=1}^{k-1} \mathbf{Prob}(n_i \text{ is a } \rightarrow\text{maximum}).$$

Finally, the probability that all of $n_1, \dots, n_k$ are $\rightarrow$maxima is obtained by multiplying this conditional probability by the probability that n_k is a $\rightarrow$maximum, so we get the required equality. □

3. Chunking Methods and Slack

3.1. **Chunking Methods and Locality.**

We begin our discussion of content-dependent chunking methods for remote differential compression by defining what we mean by a chunking method. Our definition is general enough to also cover content-independent methods, such as chopping a file into chunks of a fixed length, but we shall use it only in the content-dependent case.

Definition 18. A *chunking method* is an operation assigning to every file a set of locations in that file, called the *cutpoints* of the file. That is, for files in PFE^I where I is $[0, l-1]$ or $\mathbb{N}$ or $\mathbb{Z}$, the cutpoints form a subset of I. The *chunks* of a file F are the segments beginning at one cutpoint and ending immediately before the next cutpoint, as well as the segment from the beginning of the file (when there is a beginning, i.e., when $I \neq \mathbb{Z}$) up to and not including the first cutpoint and the segment from the last cutpoint (if there is one) to the end of the file. In the degenerate case where the set of cutpoints is empty, the whole file counts as a single chunk.

We have adopted here the arbitrary convention that a cutpoint belongs to the chunk on its right rather than the one on its left. So the actual cutting occurs just to the left of the cutpoints.

When the files are infinite (I=$\mathbb{N}$ or $\mathbb{Z}$), infinite chunks can occur. In all the chunking methods that we consider, however, the probability of such an occurrence is zero. That is, almost all files will be chopped into finite chunks. We shall often ignore the measure-zero set of exceptions. Indeed, we have already ignored it in our rough description of protocols, where the first two steps involve applying a hash function to each chunk.

Definition 19. Let h be a non-negative integer, and let $F \in \mathrm{PFE}^I$ be a file. A position $i \in I$ is *h-internal* to the file F if the interval $[i-h, i+h]$ is included in I. In this case, the restriction of F to this interval, consisting of the $2h+1$ entries $F(i-h), \dots, F(i+h)$ of F, is called the *h-vicinity* of i in F. We may omit the prefix h when it is clear from the context.

One of the advantages of dealing with doubly infinite files is that all positions are internal. In a singly infinite file, all except the first h positions are h-internal; in a finite file, the exceptions are the first and last h positions.

We shall need to compare vicinities at different positions, and for this purpose it is useful to have a brief expression for "being the same except that the positions have been shifted".

Definition 20. Consider two finite sequences of potential file entries, of the same length, but indexed by (possibly) different segments of $\mathbb{Z}$, say $\sigma \in \mathrm{PFE}^{[a,a+l]}$ and $\tau \in \mathrm{PFE}^{[b,b+l]}$. We say that σ and τ *agree* if they differ only by shifting the indices from a to b, i.e., if $\sigma(a+i) = \tau(b+i)$ for $i = 0, \dots, l$.

Definition 21. A chunking method is *local* if there exist a non-negative integer h and a nonempty set C of sequences of length $2h+1$, $C \subseteq \mathrm{PFE}^{[-h,h]}$ such that the cutpoints of any file F are exactly those h-internal positions in F whose h-vicinity agrees with some $\sigma \in C$. We call h the *horizon* of the method and C its *criterion* (for cutpoints).

The requirement that C be nonempty avoids trivialities; if it were violated, no file would have cutpoints. With this requirement, not only do some files have cutpoints, but almost all infinite files in $\mathrm{PFE}^{\mathbb{N}}$ have infinitely many cutpoints and almost all doubly infinite files in $\mathrm{PFE}^{\mathbb{Z}}$ have infinitely many positive and infinitely many negative cutpoints. Thus, almost all files are chopped into finite chunks.

Definition 22. A chunking method is *shift-invariant* if, whenever i is a cutpoint of a doubly infinite file F and s is an arbitrary integer, then $i-s$ is a cutpoint of $\mathrm{BS}^s(F)$.

It follows immediately from the definition of locality that any local chunking method is shift-invariant, simply because the vicinity of $i-s$ in $\mathrm{BS}^s(F)$ agrees with that of i in F.

3.2. **Length of Chunks.** Fix a shift-invariant chunking method for doubly infinite files. Because of shift-invariance, each position $i \in \mathbb{Z}$ has the same probability p of being a cutpoint. We call this p the *cutpoint probability* of the method. In the case of local chunking methods, we have, with notation as in the definition of locality,

$$p = \frac{|C|}{|\mathrm{PFE}|^{2h+1}}.$$

In the case of singly infinite files, locality requires all cutpoints to be h-internal, i.e., to be positions $\geq h$. All these positions have the same probability of being cutpoints, and that probability is given by the same formula as for the doubly infinite case. Similarly, for finite files, the same formula gives the probability that any particular h-internal position will be a cutpoint.

Definition 23. For a file F with a cutpoint at 0, we define the *chunk length* $L(F)$ to be the first positive cutpoint.

Thus, the chunk length of F is the length of the chunk whose first element is 0. We shall comment later, in Remark 28, on why we consider only files with a cutpoint at 0, rather than extending the definition to arbitrary files.

It is possible for a file to have a cutpoint at 0 but no cutpoints farther to the right, so that the chunk length is not defined. But, with all local chunking methods and indeed with all

shift-invariant methods that have non-zero cutpoint probability, the Poincaré recurrence theorem ensures that the files with no chunk length form a set of measure zero, so they will not affect any of the considerations below.

For finite files, on the other hand, there is a non-zero probability that the chunk length doesn't exist. But this probability approaches zero exponentially fast as the length of the file increases. So for very long files, there is only a negligible danger that the chunk length is undefined. It would also be reasonable to modify the definition of chunk length, to cover the case where a finite file $F \in \mathrm{PFE}^{[0,l-1]}$ has no positive cutpoint, by letting the chunk length in this case be the whole length l of the file.

Notation 24. Given a shift-invariant chunking method, we write p for the cutpoint probability. If a file F with a cutpoint at 0 is also given, then we write $L(F)$ for its chunk length. Thus, L is a random variable (defined almost everywhere) on the subspace Cut0 of $\mathrm{PFE}^{\mathbb{Z}}$ consisting of files with a cutpoint at 0.

Because the random variable L is defined only on the subspace Cut0, any statement about its statistical properties must be interpreted as conditional on the event Cut0. Nevertheless, to reduce the chance of confusion, we shall often (as in the next proposition) indicate the conditioning explicitly.

Proposition 25. *For doubly infinite files, the expectation of the chunk length for any shift-invariant chunking method is the reciprocal of the cutpoint probability,* $\mathbf{E}(L|\mathit{Cut0}) = 1/p$.

Proof. Notice that the first positive cutpoint of F is, by shift-invariance, the smallest positive number k such that 0 is a cutpoint of the shifted file $\mathrm{BS}^k(F)$, where BS denotes the leftward Bernoulli shift as above. Thus the expectation of the chunk length is the conditional expectation, conditioned on $F \in \mathrm{Cut0}$, of the smallest positive k with $\mathrm{BS}^k(F) \in \mathrm{Cut0}$. This is precisely the situation covered by Kac's theorem, Proposition 10. According to that theorem, the expectation under consideration is the reciprocal of the probability of Cut0, so it is $1/p$. □

Can one associate chunk lengths to (almost) all $F \in \mathrm{PFE}^{\mathbb{Z}}$, rather than only to those with a cutpoint at 0? The answer is yes, to some extent, but the right approach is more complicated than one might guess. It involves averaging over all the chunks within the file, as follows.

Definition 26. The *average chunk length* of a file $F \in \mathrm{PFE}^{\mathbb{Z}}$ is the limit

$$\lim_{N\to\infty} \frac{2N+1}{\text{Number of cutpoints in } [-N,N]}.$$

This limit and thus the average chunk length may be undefined for some files F, but these form a set of measure zero. In fact, we have the following consequence of Birkhoff's ergodic theorem.

Proposition 27. *For almost all files in* $\mathrm{PFE}^{\mathbb{Z}}$*, the average chunk length equals* $1/p$.

Proof. Before starting the main part of the proof, we note that the definition of average chunk length would be unaffected if we replaced $2N+1$ by $2N$ in the numerator (without changing the denominator), because $\frac{2N}{2N+1} \to 1$ as $N \to \infty$. We refer to such a change, also in other similar situations, as a "trivial modification" of the fraction.

Applying Birkhoff's result, Proposition 9, to the space $\mathrm{PFE}^{\mathbb{Z}}$, the leftward Bernoulli shift, and the event Cut0, and recalling that the cutpoints of a file F are exactly those k for which $\mathrm{BS}^k(F) \in \mathrm{Cut0}$, we find that

$$\lim_{N\to\infty} \frac{\text{Number of cutpoints in } [0,N]}{N} = p,$$

where we have made a trivial modification to get N rather than $N+1$ in the denominator. Symmetrically, using BS^{-1} and shifting the file by one unit (and not needing a trivial modification this time), we get

$$\lim_{N\to\infty} \frac{\text{Number of cutpoints in } [-N,1]}{N} = p.$$

Average these two equations to get

$$\lim_{N\to\infty} \frac{\text{Number of cutpoints in } [-N,N]}{2N} = p.$$

Finally, take reciprocals of both sides and make a trivial modification to get the desired result. □

Thus, Proposition 25 would remain true if we replaced chunk length by average chunk length and took the expectation over all of $\mathrm{PFE}^{\mathbb{Z}}$ rather than over Cut0. Notice, however, that average chunk length cannot replace chunk length in other contexts. For example, in a non-trivial chunking method, the chunk length will have non-zero variance, essentially because not all chunks have the same length. But the average chunk length is, according to Proposition 27, constant almost everywhere, so its variance is 0.

Remark 28. It is tempting to associate, to (almost) every file $F \in \mathrm{PFE}^{\mathbb{Z}}$, the length of a particular, chosen chunk to serve as the chunk length of F. Such a definition would avoid both the restriction to Cut0 in our definition of chunk length and the limiting process in our definition of average chunk length. In fact, an earlier draft of this paper defined the chunk length of F to be the distance between the first two non-negative cutpoints. Unfortunately, the analog of Proposition 25, averaging this chunk length over all files, is in general false, even for local chunking methods. Here is a simple counterexample. Suppose $\mathrm{PFE} = \{0, 1\}$, and let the chunking method put cutpoints at both of the 1's wherever the pattern 011 occurs in a file. (Formally, this is a local chunking method with horizon $h = 2$ and criterion C consisting of the 8 sequences $*011*$ and $011**$, where the stars represent either 0 or 1 independently.) The cutpoint probability is 1/4, and the expectation of the chunk length, as we have defined it, is 4. But the older definition, using the first two non-negative cutpoints, results in the expectation of the chunk length being only 7/4.

One can see, intuitively, what goes wrong in this example. The chunking method guarantees that cutpoints occur in adjacent pairs. So half the chunks have length 1 (extending from the first to the second 1 in a 011 pattern) while the other half are longer (of length at least 2 and on average 7). Position 0 is considerably more likely to lie in one of the long chunks, so the next chunk, the one between the first two non-negative cutpoints, is more likely to be a short one, of length 1. And of course this drags down the expectation of this version of chunk length.

Another approach to assigning a chunk length to (almost) every file is to take the length of the chunk that contains a specific position, say 0. This also fails to work properly, for similar reasons. If there are chunks of different lengths (as there will be under non-trivial chunking methods) then 0 is more likely to lie in one of the longer chunks. A specific counterexample is even easier to produce than for the "first two non-negative cutpoints" version. Let $\mathrm{PFE} = \{0, 1\}$ again and let the cutpoints be the positions where the file entry is 1. (This is a special case of the pure point filter method discussed in more detail below.) Then the cutpoint probability is 1/2 and the average chunk length is 2. But the expectation of the length of the chunk containing 0 is 3.

Typically a chunking method has parameters which can be manipulated so that the expected chunk length $\mathbf{E}(L)$ is as desired. But the chunks in a particular file may be shorter or longer than this average. As indicated earlier, both too short and too long chunks are undesirable. On the one hand, the overhead of assembling and communicating the checksum of too short a chunk may outweigh the cost of sending the chunk itself. On the other hand, excessively long chunks are unlikely to match between the sender's and recipient's files. Therefore, one would prefer a chunking method with lower deviation from the average chunk length.

In a sense, too long chunks create a smaller problem than too short chunks. If necessary, a too long chunk can be subdivided by adding additional "artificial" cutpoints; all "indigenous" cutpoints remain intact. For example, if the desired chunk length is L, then a chunk of size $cL + d$ may be subdivided into $c - 1$ chunks of size L, and one chunk of size $L + d$. In this way, a sender never transmits a chunk of size larger than $2L$. Of course, this method can run into the same problem that motivated content-dependent chunking in the first place: Since the artificial cutpoints are at fixed positions, inserting a single character into a file, near the beginning of a long chunk, may disrupt agreement between the artificial subdivisions. There are more intrinsic ways of subdividing long chunks. For example, in the case of local-maximum chunking method, discussed in section 6, a long chunk can be subdivided by means of local maxima with smaller horizon.

On the other hand, imposing a minimal length may require removing indigenous cutpoints. The following example is admittedly extreme and improbable but it gives a good idea of a discoordination that may result from removing indigenous cutpoints.

Example 29. The indigenous cutpoints partition the recipient's file F_1 into $2n$ distinct chunks $C_1, \dots, C_{2n}$ that are too small. By removing half of the cutpoints, we have n bigger chunks $C_1C_2, \dots, C_{2n-1}C_{2n}$. The sender's file F_2 was obtained from F_1 by moving C_1 to the end. The indigenous cutpoints partition F_2 into $2n$ chunks $C_2, \dots, C_{2n}, C_1$. By removing half of the cutpoints, we get bigger chunks $C_2C_3, \dots, C_{2n}C_1$. But none of these bigger chunks occurs in F_1.

Lemma 30. *No local chunking method can have an absolute guarantee that the chunks are not too short and not too long.*

Proof. Consider a doubly infinite file F with the same entry in every position. Let h be the horizon of the chunking method. Every position of F has the same h-vicinity. It follows that either every position of F is a cutpoint, which violates the minimality requirement, or else no position of F is a cutpoint, which violates the maximality requirement. □

Although files of the sort used in this proof form a set of measure zero, there is non-zero (albeit small) probability for a file to have a very long finite stretch of identical entries. In such files one will have either very short chunks (of length one) or a very long chunk (at least as long as the stretch of identical entries minus twice the horizon).

The methods that we propose in Sections 5 and 6 below provide absolute lower bounds on the chunk lengths. Upper bounds and stricter lower bounds hold with high probability. In practice, absolute guarantees are not crucial; high probabilistic guarantees are almost as good.

3.3. **Costs.** How can one compare the efficiency of different chunking methods? The cost of executing a remote differential compression protocol has several components, including

(1) The number of bytes sent over the wire (in each direction),
(2) The number of communication rounds.
(3) The cumulative time of hard-disk accesses.
(4) The computation complexity of finding the cutpoints.

In this section we concentrate on the first component. It is about minimizing the bandwidth used by a file transfer. Components 2 and 3 are highly relevant as well, but they do not depend on particular chunking methods. In particular, for any protocol that fits the rough outline in Section 1, the number of communication rounds is three, namely steps 3, 5, and 6 of that outline. (The number of communication rounds would increase to $2n + 1$ if a protocol is used recursively to depth n, as described in Remark 6. So one should, when using recursion in this way, keep in mind the trade-off between the benefit in Component 1 and the cost in Component 2.)

Component 3 can be reduced by using multiple disks and carefully laying out data on disks. A more detailed discussion of components 2 and 3 is however outside the scope of this paper. Component 4 is addressed in subsequent sections in conjunction with the particular chunking methods. In particular, we show in Section 8 that local maxima can be found efficiently.

We now turn to our primary topic, Component 1. What is sent over the wire in order to transfer a file F_2 from a sender to a recipient?

(S1) The chunk checksums sent to the recipient. Suppose that the chunks are $B_1, \dots, B_n$ and let $\mathbf{E}(L)$ be the expected chunk length. So n is usually close to $|F_2|/\,\mathbf{E}(L)$. All checksums have the same length Chk. So the number of bytes sent is

$$n \cdot \mathrm{Chk} \approx (|F_2|/\,\mathbf{E}(L)) \cdot \mathrm{Chk}$$

(S2) The indication, from the recipient, which of the chunks he wants to receive; the requested chunks are the chunks of F_2 that are not chunks of F_1.
(S3) The requested chunks B_j sent to the recipient. If k is the number of F_2 chunks that are also F_1 chunks then the number of bytes sent is

$$\sum_{B_j \text{ is wanted}} |B_j| \approx (n - k) \cdot \mathbf{E}(L).$$

The estimate in (S3) is necessarily a rough approximation because it assumes that the average length of requested chunks is the same as the average length $\mathbf{E}(L)$ of chunks in general. In fact, shorter chunks usually have a better chance of matching than long ones do. For an extreme example, suppose F_2 is obtained from F_1 by modifying every $\mathbf{E}(L)^{\text{th}}$ symbol. Then no chunks of length more than $\mathbf{E}(L)$ will match, and only chunks of length smaller than $\mathbf{E}(L)$ will ever have a chance to match. In such a case the number of requested bytes will be larger than $(n-k)\cdot\mathbf{E}(L)$. Fortunately, for the purpose of the following discussion, the important part of the formula in (S3) is not the questionable factor $\mathbf{E}(L)$ but the factor $n-k$ which indicates that we should aim for large k.

(S1) and (S2) do not depend much on the chunking method. (S1) just depends, as indicated above, on the average chunk length and the size of the checksums, both of which can be chosen independently of the choice of chunking method.

(S2) is negligible compared to (S1) + (S3).

As indicated earlier, (S1) + (S3) can be reduced by applying chunking methods recursively; see Remark 6. We shall, however, analyze and compare content-dependent chunking protocols in a simple, non-recursive context. The comparisons carry over to the corresponding recursive versions, as the benefits of recursion are essentially independent of the benefits of choosing a good chunking method.

To minimize (S3) we would like to maximize the number k of common chunks. Consider a maximal interval I common to the two files F_1 and F_2, and assume that I is long enough to contain at least one common chunk. I has the form $S\cdot C_1\cdots C_n\cdot S'$ where $C_1,\dots,C_n$ are chunks, S is a proper final segment of the preceding chunk, and S' is a proper initial segment of the subsequent chunk. The parts S and S' of the agreement interval I are wasted in the sense that the agreement of these segments of the two files doesn't reduce the transmissions needed in (S3). If I had not included any common chunks then the whole I would have been wasted.

Thus, the efficiency of a chunking method depends, in large part, on its ability to keep the wasted agreements, the segments S and S', small. In the next subsection, we introduce a mathematically convenient way to assess this ability. In that discussion, we also take into account that, although the interval I is common to the two files, its subdivision into chunks may be different near the ends. This is because whether a position is a cutpoint depends on its h-vicinity, and that may extend beyond I.

3.4. **Slack from the Left.** We introduce an idealized model of what happens near the beginning of an interval of agreement between two files. The model uses two doubly infinite files $F_1,F_2\in\text{PFE}^{\mathbb{Z}}$ that coincide at all non-negative positions but are independent elsewhere. We write F^+ for their common non-negative part (an element of $\text{PFE}^{\mathbb{N}}$) and F_1^- and F_2^- for their respective negative parts. Thus, we envision a pair of doubly infinite files that start out independent but at some point (position 0) merge and are identical thereafter.

Formally, we work with the probability space $\text{PFE}^{\mathbb{M}}$ where $\mathbb{M}$ (the symbol stands for "merge") consists of all the non-negative integers and all pairs $(i,1)$ and $(i,2)$ for negative integers i. So $\mathbb{M}=\big((\mathbb{Z}-\mathbb{N})\times\{1,2\}\big)\cup\mathbb{N}$. As with previous probability spaces, we use the product measure derived from the uniform measure on PFE. Thus, the entries in a random file $F\in\text{PFE}^{\mathbb{M}}$ are chosen independently and uniformly from PFE. If $F\in\text{PFE}^{\mathbb{M}}$ then F determines a merging pair of files (F_1,F_2) by

$$F_k(i)=\begin{cases}F(i,k) & \text{if } i<0\\ F(i) & \text{if } i\geq 0.\end{cases}$$

The extraction of the two files F_k from F amounts to a pair of projection functions

$$\pi_k:\text{PFE}^{\mathbb{M}}\to\text{PFE}^{\mathbb{Z}}:F\mapsto F_k.$$

Both of these projection functions are clearly measure-preserving; that is, $\mathbf{Prob}(\pi_k{}^{-1}(A)) = \mathbf{Prob}(A)$ for all events $A\subseteq\text{PFE}^{\mathbb{Z}}$.

Remark 31. We think of position 0 as where the agreement between F_1 and F_2 begins. Strictly speaking, 0 is where agreement begins to be enforced by the definition of the model. It is possible for the files to already have the same entry "accidentally" at -1; this happens with probability $1/|\text{PFE}|$. And the actual interval of agreement may begin even earlier, though with even smaller

probability. Modifying the model to prohibit such coincidences would introduce additional cases into our computations without significantly changing the results. So we abstain from such a modification and use the model as presented above.

Although F_1 and F_2 are infinite, they are intended to serve as mathematically convenient models for the behavior of a pair of real, finite files with an interval I of agreement as in the discussion at the end of Section 3.3. More precisely, they model the behavior near the beginning of I, where the files agree to the right and are (as a mathematical idealization) independent to the left. As indicated earlier, we would like our chunking method to minimize the wasted part S of the interval of agreement. We also want to minimize the wasted part S' at the other end of I, but that can be treated almost symmetrically (see Section 3.5 below), so we concentrate, for now, on S.

In our model, the wasted part extends at least from the merge point 0 to the first non-negative position that is a common cutpoint of both files. (It may extend further, if, after the first common cutpoint, the files have different cutpoints. This will not happen with any of the chunking methods we consider.) That motivates the following definition.

Definition 32. Let a chunking method for doubly infinite files be given. The *slack* of any $F \in \mathrm{PFE}^{\mathbb{M}}$, written $\mathrm{Slack}(F)$, is the smallest non-negative integer i that is a cutpoint of both of the derived files F_1 and F_2.

Since the slack gives information about the behavior of a chunking method when one enters an interval of agreement from the left end, we may refer to it as the $\rightarrow$slack, especially if we need to contrast it with the analogous $\leftarrow$slack defined below.

As with some previous definitions, we confess that the slack may not be defined for some F, if the files F_1 and F_2 have no common cutpoint. For local chunking methods, the set of such bad F has probability zero and can therefore safely be ignored. Indeed, if h is the horizon of the chunking method, then any cutpoint $\geq h$ in either F_1 or F_2 is also a cutpoint of the other, since the h-vicinities agree. So the only way for F to have undefined slack is for each F_k to have no cutpoints $\geq h$. But we already saw that almost all files in $\mathrm{PFE}^{\mathbb{Z}}$ have infinitely many positive cutpoints, so, invoking the fact that π_k preserves measure, we conclude that almost all $F \in \mathrm{PFE}^{\mathbb{M}}$ have well-defined slack.

For non-local chunking methods, it is not so clear that the slack is almost everywhere defined, but this will be the case for the one non-local method that we shall analyze and compare with our local methods.

It is intuitively plausible that chunking methods with large chunks will have larger slack, because chunks that start in the independent negative parts F_k^- of the two files will extend farther into the common positive part F^+. Accordingly, it makes sense to measure slack relative to expected chunk length.

Definition 33. The *normalized slack* of $F \in \mathrm{PFE}^{\mathbb{M}}$ is defined as $S(F) = \mathrm{Slack}(F)/\,\mathbf{E}(L)$.

As indicated by our discussion of (S3) above, remote differential compression benefits from a chunking method with small slack. Accordingly, we shall use the expectation of the (normalized) slack as one measure of the quality of chunking methods.

3.5. **Slack from the Right.** Recall that it is advantageous for a chunking method to waste as little as possible from either end of an interval of agreement. That is, if two files coincide on a long interval I, then the chunking method should produce a common cutpoint near the left end and another common cutpoint near the right end of I. The slack measures how well a method does at the left end. The situation at the right end is almost but not quite symmetrical. For mathematical simplicity, we make the definitions exactly symmetrical. Afterward, we discuss how reality deviates slightly from this perfect symmetry.

Symmetrical to $\mathbb{M}$ is the index set

$$\mathbb{D} = \{i \in \mathbb{Z} : i \leq 0\} \cup \{(i,k) \in \mathbb{Z} \times \{1,2\} : i > 0\}.$$

(The symbol stands for "diverge".) An element F of $\mathrm{PFE}^{\mathbb{D}}$ amounts to two files $F_k \in \mathrm{PFE}^{\mathbb{Z}}$

$$F_k(i) = \begin{cases} F(i) & \text{if } i \leq 0 \\ F(i,k) & \text{if } i > 0 \end{cases}$$

that coincide at non-positive locations but are independent at positive locations. The two projections

$$\pi_k : \mathrm{PFE}^{\mathbb{D}} \to \mathrm{PFE}^{\mathbb{Z}} : F \mapsto F_k$$

preserve measure just as before. Symmetrically to the earlier notation, we write F^- for the common part of F_1 and F_2, i.e., the restriction of F to non-positive integers, and we write F_k^+ for the independent positive parts of the two files F_k.

Definition 34. Let a chunking method for doubly infinite files be given. The *reverse slack*, or *←slack*, of any $F \in \mathrm{PFE}^{\mathbb{D}}$, which is written $\leftarrow\mathrm{Slack}(F)$, is the smallest non-negative integer i such that $-i$ is a cutpoint of both of the derived files F_1 and F_2.

The two files F_k extracted from any $F \in \mathrm{PFE}^{\mathbb{D}}$ agree at all positions ≤ 0, and the ←slack of F measures how much of this agreement is wasted. This is intended to model what happens at the right end of a long interval of agreement between two real files.

Remark 35. As a measure of wasted agreement, the ←slack suffers from a few small inaccuracies. One was already pointed out in Remark 31 in connection with the →slack, namely that there is a slight chance that the interval of agreement between F_1 and F_2 is actually longer than what the model enforces. Even though $F_1(1)$ and $F_2(1)$ are independent, they might happen to coincide. As with →slack, we choose to ignore this problem because it is unlikely to occur at all and even more unlikely to have a significant influence (more than one or two positions) on the amount of wasted agreement.

Two other inaccuracies arise from our convention that a cutpoint is included in the chunk to its right, not the one to its left. Suppose the ←slack of $F \in \mathrm{PFE}^{\mathbb{D}}$ is s. So the two files F_1 and F_2 have a common cutpoint at $-s$ and no later common cutpoints ≤ 0. Ordinarily (but see the exception in the next paragraph), this means that the chunks of F_1 and F_2 that begin at $-s$ will differ, either because they have different lengths or, if they have the same length, because they extend to positive positions where the files differ. (Remember that, as discussed above, we are ignoring possible accidental agreement at position 1.) So the agreement of F_1 and F_2 at the $s+1$ positions $-s, \dots, 0$ is wasted. The ←slack s underestimates the waste by 1. Thus, were it not for the next paragraph, this inaccuracy could be corrected by simply adding 1 to the ←slack. In most situations, this correction will be negligible; the slack is comparable to the average chunk length, which is much larger than 1.

There is, however, an exceptional situation where s overestimates the waste. This occurs when both F_1 and F_2 have a cutpoint at 1. Being > 0, this cutpoint has no influence on the ←slack s, but the chunk that begins at $-s$ is $[-s, 0]$, which lies entirely in the interval of agreement of the two files. So there is no waste at all in this case. The correction needed in this case is (to not add 1 as in the preceding paragraph and) to subtract s, i.e., to replace the ←slack with 0. This correction, though it may be large for an individual file, is usually negligible on average, especially in comparison with the whole ←slack s, because the probability q that 1 is a cutpoint in both F_1 and F_2 is so small (clearly $q \leq p$ and usually $q \ll p$).

Because all the inaccuracies in the reverse slack are relatively small, we shall neglect them and use the average reverse slack as a measure of a chunking method's waste of agreement at the right end of an interval on which two files coincide.

3.6. **Quality of Chunking Methods.** Recall the three desiderata for a chunking method: The chunks should not be too short. The chunks should not be too long. And agreements between parts of files should be promptly reflected in agreements between chunks. Of course, these desiderata are interrelated. For example, the trouble with excessively long chunks is that a long interval of agreement between files might not contain any whole chunks. Nevertheless, it is convenient to consider the three desiderata separately, because the first two are somewhat easier to deal with. We must not, however, be so focused on the first two that we ignore the third, because the first two can be satisfied by using chunks of a single fixed length, and we have seen that this chunking method can take two nearly identical files (differing by the addition of a single character) and produce no agreements at all between chunks.

The chunking methods that we consider will have one or two adjustable parameters, so that we can control, for example, the average chunk size, or the minimum chunk size, or sometimes both.

So it is not too difficult to achieve the first or second desideratum; it is the interplay between the two that imposes a non-trivial requirement on a chunking method. This interplay can be summarized by saying that we do not want too much variation in the chunk lengths.

An obvious measure, therefore, is the variance (or its square root, the standard deviation) of the chunk length. Another possible measure for the same purpose is the ratio of the average to the minimum chunk length. A third is the probability of finding no cutpoints in a long interval, say an interval whose length is 5 or more times the average chunk length. We shall calculate or at least estimate these three measures for all of the chunking methods that we treat in the following sections, except that the variance of the local maximum method remains an open problem and the probability of long chunks and the $\leftarrow$slack of the interval filter are expressed not explicitly but in terms of the smallest root of a certain polynomial equation.

The quality of a chunking method from the viewpoint of the third desideratum is, we propose, reasonably measured by the slack and reverse slack. We shall therefore also compute or at least estimate the $\rightarrow$slack and $\leftarrow$slack of the methods in the following sections, except that we have not been able to accurately estimate the slack of the local maximum method (in either direction — they are the same by symmetry).

4. Point Filter Methods

4.1. **Pure Point Filters.** The pure point-filter chunking is the most local chunking method: the vicinity of a position i that determines whether i is a cutpoint of F consists of $F(i)$ alone. This chunking method has an integer $c \geq 2$ as a parameter. In the following discussion, we consider c as fixed and we identify PFE with the set of integers from 0 to $|\mathrm{PFE}| - 1$.

Definition 36. *Pure point filter chunking* is the chunking method where the cutpoints of a file F are those positions i where $F(i)$ is divisible by c.

In practice, the number $|\mathrm{PFE}|$ of potential file entries (often hashes resulting from a rolling hash) is usually a power of 2 and c is a smaller power of 2. This simplifies the task of finding cutpoints, because, instead of dividing $F(i)$ by c, one can just test whether the bit-pattern of $F(i)$ ends with enough 0's.

When $|\mathrm{PFE}| \geq c$ are powers of 2, or more generally when c divides $|\mathrm{PFE}|$, then the cutpoint probability is clearly $1/c$. In general, it is

$$p = \frac{1}{|\mathrm{PFE}|}\left\lceil \frac{|\mathrm{PFE}|}{c} \right\rceil,$$

i.e., $1/c$ rounded up to the next larger multiple of $1/|\mathrm{PFE}|$. (Had we chosen PFE to start with 1 rather than 0, then we would round down rather than up.) In practice, the rounding is negligible because $|\mathrm{PFE}|$ is much larger than c.

Remark 37. It is only a matter of practical convenience that the cutpoints are defined in terms of divisibility. In practice, one uses divisibility by powers of two, because that can be checked by inspecting a bit pattern, which takes just one CPU cycle. But our analysis applies equally well to less efficient criteria. One could use an arbitrary subset C of PFE, defining the cutpoints of a file F to be those positions i where the value $F(i)$ belongs to C. Then the cutpoint probability is $p = |C|/|\mathrm{PFE}|$. All the following results, except for approximations involving c, hold in this more general situation.

The probability distribution of the chunk length L is geometric with parameter p. That is, L takes each positive integer i as a value with probability

$$\mathbf{Prob}(L = i) = (1-p)^{i-1}p \qquad \text{for } i \geq 1.$$

(Recall that we defined chunk length only for files with a cutpoint at 0, so the probabilistic notions here are all conditional on the event Cut0.) Using the formulas in Subsection 2.3 to sum the relevant series, we obtain that the expectation of L is

$$\mathbf{E}(L) = \frac{1}{p} \approx c$$

in agreement with Proposition 25, that

$$\mathbf{E}(L^2) = \frac{2-p}{p^2},$$

and therefore that the variance of L is

$$\mathbf{Var}(L) = \mathbf{E}(L^2) - E(L)^2 = \frac{1-p}{p^2} \approx c^2 - c.$$

The slack is also geometrically distributed, but taking values starting with 0 rather than 1. (The slack of $F \in \mathrm{PFE}^{\mathbb{M}}$ can be zero; the chunk length of $F \in \mathrm{PFE}^{\mathbb{Z}}$ cannot.) We have

$$\mathbf{Prob}(\text{Slack} = i) = (1-p)^i p \qquad \text{for } i \geq 0$$

and therefore

$$\mathbf{E}(\text{Slack}) = \frac{1}{p} - 1 \approx c - 1.$$

The expectation of the normalized slack is thus $1 - p \approx 1 - \frac{1}{c}$. Of course, the reverse slack has the same expectation, because of the left-right symmetry of the chunking method.

4.2. **Point Filters Without Short Chunks.** The pure point filter method allows chunks to be as small as a single element of a file. To avoid excessively small chunks, a modification of the method was proposed, in [20], forcing all chunks to be larger than a certain length h; we refer to this modification as the LBFS chunking method. In [20], the parameters were chosen to be $h = 2^{11}$ and $c = 2^{13}$, but the chunking method can be applied with any desired h and c. It proceeds as follows, given a file in $\mathrm{PFE}^{[0,l-1]}$ or in $\mathrm{PFE}^{\mathbb{N}}$. Ignore the first $h+1$ positions (0 to h) because cutting there would produce an impermissibly small chunk. Beginning at position $h+1$, look for positions i where c divides $F(i)$, and declare the first such i to be a cutpoint. Then ignore the next h positions, $i+1$ to $i+h$, again because cutting there would produce an impermissibly small chunk. Starting at position $i+h+1$, look again for a position where the entry in F is divisible by c, declare it to be a cutpoint, and so forth.

Because this chunking method was introduced as a part of LBFS (low bandwidth file system) in [20], we shall refer to it as *LBFS chunking*. (LBFS includes other aspects in addition to chunking, such as maintaining a system-wide database of chunks indexed by their hashes (see remark 5), but we are concerned in this paper only with the chunking method.)

Definition 38. In connection with LBFS chunking applied to a file F, we call a position i a *candidate* if c divides $F(i)$. The *cutpoints* are those candidates i such that $i > h$ and none of the preceding h positions $i-h, \dots, i-1$ is a cutpoint. If a candidate i fails to be a cutpoint because there was a cutpoint j in the range $i - h \leq j \leq i-1$, then we say that j *blocks* i (from being a cutpoint). We use the symbol k for the ratio h/c.

Notice that the candidates for the LBFS chunking method are exactly the cutpoints for the pure point filter method with the same c.

We shall occasionally give particular attention to the case $k = 1/4$ that corresponds to the choice of parameters proposed in [20].

Remark 39. The LBFS chunking method is not local (except, of course, when $h = 0$ and it reduces to the pure point filter method). To see this, consider how to tell whether a position i is a cutpoint. First, check whether c divides $F(i)$. If not, then you have the answer, "no." But if c does divide $F(i)$, then you still have to check whether any of $i-1, \dots, i-h$ was a cutpoint, which would block i. So check whether c divides any of $F(i-1), \dots, F(i-h)$. If all the answers are "no", then you have the answer; i is a cutpoint. But if you find a candidate among $i-1, \dots, i-h$, then you still need to check whether it was a cutpoint, which involves checking the preceding h positions. And if you find a candidate there, then you have to check h positions farther back from that candidate, and so forth. Thus, there is no a priori bound on how far back you might have to look in order to decide whether i is a cutpoint. That is, the chunking method is not local.

We observed earlier that, for any local chunking method, almost every infinite file $F \in \mathrm{PFE}^{\mathbb{N}}$ has infinitely many cutpoints. Since the LBFS method is not local, this observation cannot be applied to it directly. Nevertheless, one can easily deduce the desired information, that the LBFS

method produces infinitely many cutpoints in almost all $F \in \mathrm{PFE}^{\mathbb{N}}$, from the corresponding result for the pure point filter method, which is local. To see this, consider an arbitrary position $i \in \mathbb{N}$. For almost all F, the pure point filter method will have a cutpoint $j > i + h$. This j is a candidate in the LBFS method, so either it is a cutpoint or it is blocked by some cutpoint in the interval from $j - h$ to $j - 1$. In either case, there is a cutpoint $> i$, either j or the one blocking it. Since this holds with probability 1 for each i (and since probability is countably additive), almost all F have infinitely many cutpoints under the LBFS chunking method.

Remark 40. We defined the LBFS method for finite and singly infinite files but not for doubly infinite files. If one applies the same idea to doubly infinite files, it may fail to give a well-defined set of cutpoints. That is, it may not really be a chunking method. The source of this problem is the same as the source of non-locality in the previous remark. To determine whether a position i is a cutpoint, we may have to look farther and farther back in the file. In the case of doubly infinite files, the process may never terminate, so there is no decision whether i is a cutpoint.

Consider, for example, a file $F \in \mathrm{PFE}^{\mathbb{Z}}$ such that $F(i)$ is divisible by c if and only if i is divisible by h. So every h^{th} position is a candidate, but each such position, if it is a cutpoint, will block the next candidate from being a cutpoint. It would be consistent with the LBFS method to say that the cutpoints are all of the positions divisible by $2h$, i.e., every second candidate. Each of these candidates $2nh$ blocks $(2n+1)h$, but then $(2n+2)h$ is unblocked and serves as the next cutpoint. But it would be equally consistent to say that the cutpoints are the positions of the form $(2n + 1)h$, the odd multiples of h. Each blocks the next even multiple of h, and then the next odd multiple of h is unblocked and serves as the next cutpoint. The LBFS method gives no way to choose between these two possible selections of cutpoints from among the candidates. One could amplify the method by specifying the choice arbitrarily in all such situations; the result would be a chunking method, but it would not be shift invariant.

Fortunately, the LBFS method works for almost all doubly infinite files. To see this, notice first that, if a file has no candidates in some interval of length h, then the method will determine the cutpoints to the right of that interval. Specifically, the first candidate to the right of the interval is not blocked, because there is no candidate in the preceding h positions. So this first candidate is a cutpoint. Knowing this, one can proceed to the right, inductively determining which candidates are cutpoints and which are blocked. Furthermore, an easy calculation shows that almost all files $F \in \mathrm{PFE}^{\mathbb{Z}}$ have candidate-free intervals of length h arbitrarily far to the left. So, in almost all files, all the cutpoints are uniquely determined.

From now on, we shall work with the LBFS method as though it were a genuine chunking method even for doubly infinite files. That is, we shall ignore the measure-zero set of exceptional files for which the method fails to determine the cutpoints.

Having made the LBFS chunking method applicable to doubly infinite files, by ignoring a set of probability zero, we note that the method is clearly shift-invariant even though it is not local.

Remark 41. In the preceding remark, we used the fact that a candidate-free interval of length h is a doubly infinite file is sufficient to disambiguate the choice of cutpoints to the right. There are other intervals that would serve the same purpose. For example, if $h = 3$ and if a file contains a segment of the form NCCCCNNC, where C means "candidate" and N means "non-candidate, then the last candidate in this segment will be a cutpoint, and the chunking method to the right of this segment is therefore well-defined. To see that the last C in NCCCCNNC is a cutpoint, suppose not. Then it is blocked by a cutpoint at one of the $h = 3$ preceding positions, which can only be the last of the four consecutive C's. Then the preceding three C's are not cutpoints, lest they block the fourth one. But why is the third of the four consecutive C's not a cutpoint? It's not blocked by the two preceding C's (as they're not cutpoints), nor by the initial N (as a non-candidate is certainly not a cutpoint). This contradiction shows that the last C in NCCCCNNC must be a cutpoint. It is clearly possible to devise analogous examples, and more complicated ones, also for other values of h.

The LBFS chunking method clearly ensures that all chunks have length at least $h + 1$. The following proposition gives basic probabilistic information about the behavior of this method. The slack is more complicated and is treated in the next subsection.

For notational simplicity, we assume henceforth that |PFE| is divisible by c. In the general case, the following results are still approximately correct and would become exactly correct if c were replaced with $|\mathrm{PFE}|/\lceil |\mathrm{PFE}|/c \rceil$.

Proposition 42. *The LBFS chunking method applied to (almost all) doubly infinite files has the following properties.*

(1) *Each position is a candidate with probability $1/c$ independently.*
(2) *The expectation of the chunk length is $h + c = c(1 + k)$.*
(3) *Each point has the same probability to be a cutpoint, namely $1/(c + h)$.*
(4) *Any interval of $l \leq h + 1$ positions contains a cutpoint with probability $l/(c + h)$.*
(5) *The variance of the chunk length is $\mathbf{Var}(L) = c^2 - c$.*

Proof. Item 1 was already established in our discussion of the pure point filter method, since the cutpoints of that method are exactly the candidates of LBFS.

In item 2, the summand h represents the blocked positions immediately after a cutpoint (at 0), and c is the expected number of subsequent positions needed to reach a candidate.

That each point has the same probability of being a cutpoint is obvious by shift-invariance. The value of the probability in item 3 follows from the expected chunk length via Kac's theorem, as applied in Proposition 25.

For item 4, recall that an interval of length $\leq h+1$ can't contain more than one cutpoint. So, as i ranges over the l points in the interval, the events "i is a cutpoint" are mutually exclusive and have probability $1/(c + h)$ each.

Finally, item 5 simply says that the variance of the chunk length is the same as for the pure point filter method. Informally, this is true because waiting h steps before looking for candidates increases chunk lengths by h but doesn't affect differences between lengths. Formally, simply observe that the probability that a chunk, starting at a particular cutpoint, has length i is 0 for $i \leq h$ and $(1 - \frac{1}{c})^{i-h-1}\frac{1}{c}$ for $i > h$. That is, the probability distribution of the chunk lengths is obtained from that of the pure point filter method by shifting h steps to the right. The shift increases the expectation by h but has no effect on the variance. □

For future reference, it will be convenient to express the variance of L in terms of the cutpoint probability and the parameter h.

Corollary 43. *For the LBFS method,*

$$\mathbf{Var}(L) = \frac{1}{p^2} - \frac{2h+1}{p} + h^2 + h\ .$$

Proof. Substitute $c = \frac{1}{p} - h$ into item 5 of the proposition. □

Remark 44. The mutual exclusion used in the proof of item 4 can also be used to obtain the probability $1/(c+h)$ in item 3 without invoking Kac's theorem. Letting p be the probability that a particular position i is a cutpoint, we find that

$$p = \frac{1}{c}(1 - hp).$$

The first factor here, $1/c$, is the probability that i is a candidate. The second factor is the independent probability that it is unblocked. Indeed, for each of the h immediately preceding positions $j = i - h, \ldots, i - 1$, there is probability p that j is a cutpoint, and these events are mutually exclusive. So hp is the probability that i is blocked. Independence follows from the observation that whether j is a cutpoint depends only on the file entries at j and to the left, not on the entry at i. Solving the equation above for p, we get $1/(c + h)$.

Of course it is also possible to get the expected chunk length by a direct computation (using formulas from Section 2.3) and the probability distribution described in the proof of item 5.

Remark 45. The number of candidates skipped by the LBFS method, after it finds a cutpoint and before it begins to look for the next candidate, is binomially distributed, for it is the number of "successes" (candidates) in h independent "trials" (positions), each trial having success probability $1/c$. Thus, the expected number of skipped candidates is $h/c = k$. The probability that

at least one candidate will be skipped, i.e., that the chunk is larger than what the pure point filter method would produce, is

$$1-\left(1-\frac{1}{c}\right)^h \approx 1-e^{-k}$$

for large c. If, for example, we want the minimum chunk size $h+1$ to be about half of the average chunk size $c+h$ (as is the case for the local maximum method in Section 6), then we would have $k=1$ and so the probability that LBFS skips a candidate is approximately $1-(1/e) \approx 63\%$ — and the probability of skipping at least two candidates is $\approx 26\%$. Thus in this situation, the LBFS method and the pure point filter method differ on a large fraction of the chunks.

4.3. **The Slack of LBFS Chunking.** In this subsection, we shall estimate the expected slack of the LBFS method. Before proceeding, we must check that the notion of slack makes sense in this context. Immediately after defining slack (Definition 32), we observed that, although some $F \in \mathrm{PFE}^{\mathbb{M}}$ may fail to have a slack, because the component files F_1 and F_2 have no common non-negative cutpoint, the probability of this event is zero for any local chunking method. Since the LBFS method is not local, a separate argument is needed here, but it is an easy one. With probability 1, the common positive part F^+ of F_1 and F_2 will contain an interval of length h with no candidates and therefore no cutpoints, and it will have a candidate to the right of this interval. The first such candidate is a common cutpoint of F_1 and F_2, and so Slack(F) is defined. Since we are interested in the expected slack, we can ignore the measure 0 set of F's whose slack is undefined.

The exact value of the expected slack seems to be difficult to compute, so we shall estimate it from below.

Consider a random $F \in \mathrm{PFE}^{\mathbb{M}}$, giving rise to a merging pair of files $F_1, F_2 \in \mathrm{PFE}^{\mathbb{Z}}$. The slack depends on the location of candidates in the common part F^+ and also on any cutpoints that may be present in F_1^- and F_2^- in the *critical range* of positions $-h, -h+1, \dots, -2, -1$, the last h positions before the merge. These positions are called critical because a cutpoint there, in either file, could block a candidate in F^+ from being a cutpoint of that file and could thus delay the appearance of a common cutpoint.

With bad luck, the delay could be quite large. Suppose, for example, that F^+ has candidates at exactly the positions $i_n = hn$ for all non-negative n, and suppose further that F_1 has a cutpoint in the critical range but F_2 does not. The critical cutpoint in F_1 will block the candidate at 0, so the next cutpoint of F_1 will be at h. That will, in turn, block $2h$, so the next cutpoint is at $3h$, and so forth. In F_2, on the other hand, 0 is not blocked, so it is a cutpoint, and it blocks h. The next cutpoint in F_2 is $2h$, blocking $3h$, and so forth. Thus, the cutpoints of F_1 (resp. F_2) are hn for odd (resp. even) n, and there are no common cutpoints. (The example doesn't really depend on the assumed precise spacing of the candidates. It would suffice that the distance $i_{n+1} - i_n$ be $\leq h$ and that $i_{n+2} - i_n$ be $> h$ for all n.) As indicated above, the probability of this situation is zero, but there is a non-zero probability that this sort of alternation continues for a large (but finite) number of steps.

We must take such situations into account when estimating the slack, because the delay in getting the files to agree on a cutpoint can greatly increase the slack.

We shall take the delay into account, but only partially. In other words, we shall take too optimistic a view of the possible delay. This is why our estimate of the slack will be low. More specifically, we shall take into account that, if one or both files F_1 and F_2 have a cutpoint in the critical range, then the last of these cutpoints, say at position $-j$ (where $1 \leq j \leq h$), blocks, in at least one of the files, all candidates up to and including $h-j$. If one of the files has a cutpoint z in $[0, h-j]$ that is blocked in the other file, then there cannot be a common cutpoint until at least position $z+h+1$. We shall pretend that the next candidate after $z+h$ in F^+ is a common cutpoint of the two files and is therefore the slack. In reality, it can happen as in the situation described above that, while one of the files has its candidates up to $z+h$ blocked, the other has a cutpoint in that part of F^+, which blocks the candidate that we pretend is a common cutpoint. That is why our pretense is too optimistic and our estimate of the expected slack too low.

We define Slack$'(F)$, for almost all $F \in \mathrm{PFE}^{\mathbb{M}}$, to be

- the first candidate $> z+h$ in F^+ if $z \geq 0$ is a cutpoint of one of the files F_1 and F_2 but is blocked in the other file by a cutpoint in the critical range, and
- $\mathrm{Slack}(F)$ if there is no such z.

Thus, $\mathrm{Slack}'(F)$ is where we optimistically pretend to have the first common cutpoint of F_1 and F_2. It is our low estimate of $\mathrm{Slack}(F)$. We write S' for $\mathrm{Slack}'/(c+h)$; it is our low estimate of the normalized slack $S = \mathrm{Slack}/(c+h)$.

We compute the expectation of Slack' by dividing the probability space $\mathrm{PFE}^{\mathbb{M}}$ (or rather the measure one subset where the cutpoints of F_1 and F_2 are well defined) into several pieces P_r, computing for each piece its probability p_r and the conditional expectation e_r of Slack', and finally adding all the products $p_r e_r$ to get the overall expectation $\mathbf{E}(\mathrm{Slack}')$, our lower bound for $\mathbf{E}(\mathrm{Slack})$. There are six pieces, defined as follows; in each case, we describe the conditions on the component files F_1 and F_2 that put $F \in \mathrm{PFE}^{\mathbb{M}}$ into that piece. Remember that F_1^- and F_2^- are the independent negative parts of F_1 and F_2 while F^+ is their common positive part. Remember also that no two cutpoints of a file can be within a distance h of each other, so each of F_1 and F_2 has at most one cutpoint in the critical range.

(1) Neither F_1 nor F_2 has a cutpoint in the critical range $[-h,-1]$.
(2) Both F_1 and F_2 have a cutpoint at the same position $-j \in [-h,-1]$.
(3) F_1 has a cutpoint $-j \in [-h,-1]$ and F^+ has a candidate z that is a cutpoint of F_2 but blocked by $-j$ in F_1.
(4) F_1 has a cutpoint $-j \in [-h,-1]$, F_2 has no cutpoint in $[-j,-1]$, and there is no z as in (3).
(5)–(6) Like (3)–(4) but with the roles of F_1 and F_2 interchanged.

Before treating these pieces individually, we check that they constitute a partition of $\mathrm{PFE}^{\mathbb{M}}$ (minus, as always, the set of measure zero where the cutpoints are not well defined). It is clear that (1) is disjoint from all the others. Observe that, in (3), no point in $[-j,-1]$ can be a cutpoint of F_2, because such a cutpoint would block z from being a cutpoint of F_2. This observation and the analogous one in (5) immediately show that all six pieces are disjoint. To see that they cover (almost) all of $\mathrm{PFE}^{\mathbb{M}}$, consider any F for which the cutpoints of F_1 and F_2 are well defined. If it is not in piece (1) or (2), then at least one of F_1 and F_2 has a cutpoint in $[-h,-1]$ and, if both do, then they are not at the same point. We can thus classify F according to which of the files has its critical cutpoint farther to the right. If this is F_1, then we clearly have (3) or (4), and if it is F_2 then, symmetrically, we have (5) or (6).

Now let us compute the probabilities p_r and the conditional expectations e_r of Slack' for the six pieces in turn. Of course there are only four computations, since (5) and (6) give the same results as (3) and (4) by symmetry.

A preliminary observation will be useful in these computations. The first non-negative candidate in a random $F \in \mathrm{PFE}^{\mathbb{M}}$ is exactly the slack of F under the pure point filter method. We have already computed its expectation as $c-1$. More generally, the first candidate at a position $> i$ in a random file has expectation $c+i$. This follows from the special case already done (where i is -1) by shift invariance.

Piece 1. By Proposition 42, the probability that F_1 has a cutpoint in the critical range of length h is $h/(c+h)$. So the probability that it has no cutpoint there is $c/(c+h)$, and similarly for F_2. Since the negative parts F_1^- and F_2^- are independent, we get

$$p_1 = \left(\frac{c}{c+h}\right)^2.$$

Because there are no cutpoints in $[-h,-1]$, the first candidate in the non-negative part F^+ will be a common cutpoint of F_1 and F_2. So the conditional expectation of the slack (and Slack') is the (conditional) expectation of the first non-negative candidate. We put "conditional" in parentheses here, because the condition (1) makes no difference. The condition refers only to the negative parts of the files while the candidate we seek is determined by the non-negative parts. Thus, by the preliminary observation, we have

$$e_1 = c-1$$

and the contribution of piece (1) to the overall expectation is

$$p_1 e_1 = \frac{c^2(c-1)}{(c+h)^2}$$

Piece 2. We split piece (2) into h sub-pieces, according to the value of $j \in [1,h]$, and we compute the probabilities $p_2(j)$, the expectations $e_2(j)$, and the contributions to the overall expectation separately for the sub-pieces.

Consider therefore, a fixed $j \in [1,h]$. The probability that F_1 has a cutpoint at $-j$ is, according to Proposition 42, $1/(c+h)$, and the same for F_2. Since these events refer only to the negative parts F_1^- and F_2^-, they are independent, and so

$$p_2(j) = \frac{1}{(c+h)^2}.$$

When F_1 and F_2 have cutpoints at $-j$, these cutpoints block candidates up to and including $h-j$. So the slack (and Slack$'$) is the first candidate position $> h-j$. By the preliminary observation, this has expectation

$$e_2(j) = c + h - j.$$

The contribution of sub-piece j to the overall $\mathbf{E}(\text{Slack}')$ is the product $p_2(j)e_2(j)$, and summing over all j we get the contribution of piece (2)

$$\begin{aligned} p_2 e_2 &= \sum_{j=1}^{h} \frac{1}{(c+h)^2}(c+h-j) \\ &= \frac{h}{c+h} - \frac{1}{(c+h)^2}\frac{h(h+1)}{2} \\ &= \frac{ch + \frac{1}{2}h^2 - \frac{1}{2}h}{(c+h)^2} \\ &= \frac{h}{(c+h)^2}\left(c + \frac{h-1}{2}\right). \end{aligned}$$

We note that p_2 has the simple formula $h/(c+h)^2$, and this combines with the preceding formula for p_2e_2 to give $e_2 = c + (h-1)/2$. The latter has the intuitive interpretation that j, being random in the interval $[1,h]$ is on average $(h+1)/2$ and so the cutpoints at $-j$ in both files block candidates up to and including, on average, $(h-1)/2$. So the first common cutpoint will be the first candidate $> (h-1)/2$, and its expectation is, by the preliminary observation, $c + (h-1)/2$. Using the average j instead of considering each j individually can be justified, but the justification ultimately amounts to the computation we did above, treating each j separately and adding the resulting contributions.

Piece 3. With j and z as in (3), notice that $0 \le z \le h-j$, because z is blocked in F_1 by $-j$. Since z is a cutpoint in F_2, it blocks all candidates up to and including $h+z$. So to find a common non-negative cutpoint of F_1 and F_2, we must look for candidates in F^+ strictly to the right of $h+z$. As indicated above, we will pretend that the first such candidate is a common cutpoint; formally, this means that we deal with Slack$'(F)$ instead of Slack(F).

As in the previous subsection, we split the piece under consideration into sub-pieces, this time indexed by both j and z, where j ranges from 1 to h and z ranges from 0 to $h-j$.

We begin by computing the probability $p_3(j,z)$ of the sub-piece indexed by j and z. The probability that F_1 has a cutpoint at $-j$ is, by Proposition 42, $1/(c+h)$. The probability that F_2 has a cutpoint at z is the same. And these two events are independent because the former depends only on F_1^- while the second depends only on F_2, i.e., on F_2^- and F^+, and these parts of F are independent. We need not consider separately the requirement in (3) that z is blocked by $-j$ in F_1; this is already covered by the fact that j and z are in the appropriate ranges, specifically that $-j < z \le h-j$. So we infer that

$$p_3(j,z) = \frac{1}{(c+h)^2}.$$

The conditional expectation of Slack$'$ is the expectation of the first cutpoint of F^+ strictly after $h+z$, so, by our preliminary observation,

$$e_3(j,z) = c+h+z.$$

Thus, the contribution of this sub-piece to the overall $\mathbf{E}(\text{Slack}')$ is

$$p_3(j,z)e_3(j,z) = \frac{c+h+z}{(c+h)^2}.$$

To get the contribution of the whole piece (3) to $\mathbf{E}(\text{Slack}')$, we must sum this over z from 0 to $h-j$ and then over j from 1 to h. Since the expression being summed is independent of j, we prefer to carry out the sum over j first; converting the limits of summation to the new order, we get

$$\begin{aligned} p_3e_3 &= \sum_{z=0}^{h-1}\sum_{j=1}^{h-z} \frac{c+h+z}{(c+h)^2} \\ &= \sum_{z=0}^{h-1} \frac{(c+h+z)(h-z)}{(c+h)^2}. \end{aligned}$$

We postpone simplifying the sum over z in order to make it easier to combine with the forthcoming result from piece (4).

Piece 4. In the description of piece (4), the requirement that there is no z as in (3) is equivalent to saying that F_2 has no cutpoint in the interval $[0,h-j]$. Indeed, a cutpoint in that interval would be a candidate of F^+ and would be blocked in F_1 by $-j$, whereas a cutpoint farther to the left would not be in F^+ while one farther to the right would not be blocked by $-j$.

Combining this requirement with the other requirement in (4) that F_2 have no cutpoint in $[-j,-1]$, we find that we can restate (4) as follows: F_1 has a cutpoint $-j$ in the critical range, but F_2 has no cutpoint in the interval $[-j,h-j]$ of length $h+1$.

As before, we divide our piece into sub-pieces, according to the value of j, and we do the calculation for each j separately.

To calculate the probability $p_4(j)$ of the sub-piece indexed by j, we again apply Proposition 42. The probability that F_1 has a cutpoint at $-j$ is $1/(c+h)$. The probability that F_2 has a cutpoint in $[-j,h-j]$ is $(h+1)/(c+h)$, so the complementary probability, that it has no cutpoint in this interval, is $(c-1)/(c+h)$. Since F_2 is independent of the negative part F_1^- of F_1, it follows that we can simply multiply the two probabilities to get

$$p_4(j) = \frac{1}{c+h}\cdot\frac{c-1}{c+h} = \frac{c-1}{(c+h)^2}.$$

Notice that neither F_1 nor F_2 has a cutpoint in $[-j+1,h-j]$. We already pointed this out for F_2, with the slightly longer interval $[-j,h-j]$, but it also holds for F_1 because this file has a cutpoint at $-j$, which blocks the next h positions. Therefore, the next candidate in F^+ strictly to the right of $h-j$ will be a common cutpoint and will thus be the slack (and Slack$'$) of F. The expectation is, by our preliminary observation,

$$e_4(j) = c+h-j.$$

Therefore, the contribution of this sub-piece to $\mathbf{E}(\text{Slack}')$ is

$$p_4(j)e_4(j) = \frac{(c-1)(c+h-j)}{(c+h)^2}.$$

To get the contribution of piece (4) to $\mathbf{E}(\text{Slack}')$, we must sum over j; it is convenient to postpone actually doing this summation until after we combine the results from pieces (3) and (4), but we prepare for the job of combining these pieces by changing the summation variable from j to $z=h-j$. (In contrast to previous computations, z does not represent a position here. It is

merely a formal variable. We chose the letter z to facilitate combination with the sum from piece (3).)

$$p_4 e_4 = \sum_{j=1}^{h} \frac{(c-1)(c+h-j)}{(c+h)^2} = \sum_{z=0}^{h-1} \frac{(c-1)(c+z)}{(c+h)^2}.$$

Assembling the pieces. Since all the contributions to $\mathbf{E}(\text{Slack}')$ computed above had $(c+h)^2$ in their denominators, we put those denominators aside for the time being. That is, we assemble the contributions to $(c+h)^2\,\mathbf{E}(\text{Slack}')$.

As indicated above, we first combine the contributions from pieces (3) and (4). We get

$$\begin{aligned}\sum_{z=0}^{h-1}(c+h+z)(h-z) + \sum_{z=0}^{h-1}(c-1)(c+z) = \\ &= \sum_{z=0}^{h-1}\left(ch+h^2+c^2-c-z(z+1)\right) \\ &= ch^2+h^3+c^2h-ch-\sum_{m=1}^{h} m(m-1) \\ &= ch^2+h^3+c^2h-ch-\frac{h^3-h}{3},\end{aligned}$$

where we introduced the new summation variable $m = z+1$ and then applied one of the summation identities from Section 2.3. Simplifying slightly by combining the two h^3 terms, and doubling the result so as to account for pieces (5) and (6) as well as (3) and (4), we get as the contribution of these four pieces to $(c+h)^2\,\mathbf{E}(\text{Slack}')$

$$2ch^2 + \frac{4}{3}h^3 + 2c^2h - 2ch + \frac{2}{3}h.$$

Finally, adding the contributions $c^3 - c$ from piece (1) and $ch + \frac{1}{2}h^2 - \frac{1}{2}h$ from piece (2), and restoring the $(c+h)^2$ denominator, we get

$$\mathbf{E}(\text{Slack}') = \left(2ch^2 + \frac{4}{3}h^3 + 2c^2h - ch + \frac{1}{6}h + c^3 - c^2 + \frac{1}{2}h^2\right)/(c+h)^2.$$

Recall that $\text{Slack}'(F) \le \text{Slack}(F)$ for all F, with the usual exception of a measure zero set where the chunking method doesn't work or too few cutpoints exist. Thus, our calculation proves the following lower bound for the expected slack.

Proposition 46. *The expectation of the slack* $\mathbf{E}(\text{Slack})$ *of the LBFS method with parameters* c *and* h *is greater than or equal to*

$$\left(2ch^2 + \frac{4}{3}h^3 + 2c^2h - ch + \frac{1}{6}h + c^3 - c^2 + \frac{1}{2}h^2\right)/(c+h)^2.$$

The expectation $\mathbf{E}(S)$ of the normalized slack is the same except that the denominator $(c+h)^2$ is replaced with $(c+h)^3$, because, by Proposition 42, $c+h$ is the expectation of the chunk length.

When, as is usually the case in practice, c and h are fairly large numbers, then, since the coefficients in our formulas are not particularly large, we can approximate the formulas by keeping only the terms of highest degree.

Corollary 47. *For large* c *and* h*, the expectation of the normalized slack is asymptotically bounded below by*

$$\left(2ch^2 + \frac{4}{3}h^3 + 2c^2h + c^3\right)/(c+h)^3.$$

In terms of the parameter $k = h/c$*, this asymptotic lower bound is*

$$\left(\frac{4}{3}k^3 + 2k^2 + 2k + 1\right)/(1+k)^3.$$

In particular, for the LBFS method with $k = 1/4$ (the choice proposed in [20]), we find that the expectation of the normalized slack is asymptotically at least 0.842. Here is a table of values of the asymptotic lower bound in the corollary, as k ranges from 0 to 1 in steps of 1/10.

k	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\mathbf{E}(S)$	0.92	0.86	0.83	0.80	0.79	0.78	0.78	0.78	0.79	0.79

This function of k attains its minimum (over positive values of k) at $k = 1/\sqrt{2}$, the minimum value being

$$\frac{8}{3}\left(1 - \frac{1}{\sqrt{2}}\right) \approx 0.781049.$$

Corollary 48. *For large c and h, the expectation of the normalized slack of the LBFS method is at least 0.781.*

Remark 49. The preceding estimates do not imply that the normalized slack of the LBFS method is optimized (i.e., minimized) by choosing k to be $1/\sqrt{2}$. This choice optimizes (for large c and h) the expectation of S', but the difference between Slack$'$ and Slack, and thus also the difference between the normalizations S' and S, increase as k increases, and so we would expect the optimum of $\mathbf{E}(S)$ to occur at a value of k somewhat smaller than $1/\sqrt{2}$.

To see why the difference between Slack$'$ and Slack should grow with k, recall that Slack$'$ is what Slack becomes if we pretend that files do not have cutpoints in a certain interval whose length is, on average, about $h/2$. The probability that this pretense is an error, i.e., the probability that there is a cutpoint in such an interval, is near

$$\frac{h/2}{c+h} = \frac{k/2}{1+k},$$

which is an increasing function of k for positive k.

Remark 50. The preceding observations suggest the question of the behavior of $\mathbf{E}(S)$ as $k \to \infty$. The lower bound $\mathbf{E}(S')$ computed above approaches 4/3, but we expect $\mathbf{E}(S)$ to be significantly larger. Here is a rough argument to show that $\mathbf{E}(S)$ is approximately proportional to k^2 for large k.

Since k is large, h is much larger than c, and so the cutpoints in a random file will be spaced approximately h positions apart. Indeed, the LBFS method always skips h positions after a cutpoint, but then it will very quickly, in about c more steps, find a candidate, which will serve as the next cutpoint. Notice that the h skipped positions will usually include a large number (approximately k) of blocked candidates.

Thus, an interval of length h is very likely to contain a cutpoint, but where in the interval that cutpoint lies, or equivalently the remainder of the cutpoint modulo h, is essentially random, being determined by some event in the distant past (where there were h consecutive non-candidates or some other special configuration of file entries that makes the LBFS cutpoints well-defined thereafter) and the small (relative to h) differences between h and the actual chunk lengths.

In particular, in the model PFE$^{\mathbb{M}}$ of merging files, we expect F_1 and F_2 to have cutpoints in the critical range $[-h, -1]$, and we expect the location of these cutpoints to be uniformly randomly distributed in this range. Now when two points are chosen at random from an interval, the expectation of the smaller (resp. larger) is at the left (resp. right) trisection point of the interval. So, on average, one of our files has a cutpoint at $-2h/3$ and the other at $-h/3$. (This use of averages is one of several reasons why this is a rough argument.) The next cutpoints after these will come about h positions later, i.e., near $h/3$ and $2h/3$, respectively.

Our optimistic estimate for the slack takes into account that one file's cutpoint at $h/3$ blocks the other's cutpoint at $2h/3$ from being a common cutpoint, but it then assumes that the next cutpoint of the former file, h units past the cutpoint at $h/3$, is a common cutpoint. So it pretends to see a common cutpoint near $4h/3$. This accounts, since the chunk length is essentially h, for the value 4/3 that we found for the expectation of the normalized slack.

But the assumption that there is a common cutpoint near $4h/3$ is usually wrong; one file has a cutpoint near $2h/3$ blocking any candidate near $4h/3$. Were it not for random fluctuations, the two files would alternate cutpoints, one having cutpoints near $(n + \frac{1}{3})h$ and the other near $(n + \frac{2}{3})h$ for all n, and there would never be a common cutpoint. In reality, the positions of the

cutpoints modulo h will slowly drift as we proceed farther to the right in the file. How long will it take them to drift to a collision, i.e., to a common cutpoint?

The distance between the cutpoints is initially (i.e., just before the merge at position 0) proportional to h; on average it is $h/3$. From one cutpoint to the next, each increases, modulo h, by a random amount of size approximately c. So the distance between them increases or decreases randomly, by amounts roughly proportional to c. So the number of steps needed for the distance to become zero can be approximated by the number of steps needed for a random walk, in steps of length roughly $c = h/k$, to cover a distance proportional to h. But the distance covered by a random walk is known to be proportional to the size of the step times the square root of the number of steps. So the number of steps needed will be roughly proportional to k^2. Since each step of the random walk corresponds to advancing a distance roughly equal to the chunk length in the file, the number of steps is approximately the normalized slack. Thus, we expect the normalized slack, for large k, to be roughly proportional to k^2.

4.4. **The Reverse Slack of LBFS Chunking.** The definition of the cutpoints in LBFS chunking is not symmetrical with respect to left and right, so there is no reason to expect the $\leftarrow$slack of this chunking method to equal the $\rightarrow$slack. We therefore calculate the reverse slack here. This turns out to be easier than the preceding calculation of the $\rightarrow$slack, in that we can obtain an exact answer rather than only a lower bound. The reason the reverse slack is easier to compute is that, when we have a candidate in the common part F^- of the two files F_1 and F_2, the question whether it is a cutpoint depends only on F^-, not on the diverged, independent F_1^+ and F_2^+. We can therefore obtain simple formulas for the probability that the reverse slack of a file is s.

This probability is the product of two factors, namely the probability that $-s$ is a cutpoint (in F^-) and the conditional probability, given that $-s$ is a cutpoint, that there is no other cutpoint in the interval $[-s+1, 0]$. The first factor here is just the cutpoint probability, already computed as $1/(c+h)$ in Proposition 42. For the second factor, we must consider two cases, depending on the relative size of s and h. If $s \le h$, then the second factor is 1, because the interval $[-s+1, 0]$ is within the range where $-s$ blocks all candidates from being cutpoints. If $s > h$, however, then $-s$ blocks candidates only up to and including $-s+h$ and so we must still consider possible cutpoints in the interval $[-s+h+1, 0]$. Notice that there is a cutpoint in this interval if and only if there is a candidate there, because the first candidate there, if any, will be a cutpoint. Thus, the second factor for our computation can be expressed as the probability, conditional on a cutpoint at $-s$, of having no candidates among the $s-h$ positions in $[-s+h+1, 0]$. But the events "i is a candidate" for such positions i are independent of each other and of the cutpoint at $-s$. Each of these events has probability $1/c$, so the probability that none of them occurs is $(1-\frac{1}{c})^{s-h}$. Therefore, the probability that s is the reverse slack is, when $s > h$, given by $(1-\frac{1}{c})^{s-h}/(c+h)$.

Therefore, the average $\leftarrow$slack for the LBFS chunking method is given by

$$\sum_{s=0}^{h} s\frac{1}{c+h} + \sum_{s=h+1}^{\infty} s\frac{1}{c+h}\left(1-\frac{1}{c}\right)^{s-h}.$$

A routine calculation, using the formulas in Section 2.3, yields the explicit form of the $\leftarrow$slack:

$$\frac{h(h+1)}{2(c+h)} + c - 1 = \frac{h^2 - h + 2c^2 - 2c + 2ch}{2(c+h)}.$$

The normalized reverse slack, obtained by dividing this result by the expectation $c+h$ of the chunk length, is

$$\frac{h^2 - h + 2c^2 - 2c + 2ch}{2(c+h)^2} \approx \frac{k^2+2k+2}{2(k+1)^2} = \frac{1}{2} + \frac{1}{2(k+1)^2},$$

where k is, as before, h/c, and where the approximation is for large h and c with fixed k.

In particular, when $k = 1/4$ as in [20]), the normalized reverse slack is approximately 0.82.

If we allow k to vary, the expected normalized $\leftarrow$slack, as a function of k (for large h and c) is monotonically decreasing, but there is no use choosing a large k in order to make the $\leftarrow$slack small, for we have seen that this would make the $\rightarrow$slack large. What we can reasonably do is to compute the value of k that minimizes the sum of the normalized reverse slack and our lower

bound for the normalized slack. That will provide a lower bound for the normalized →slack plus ←slack over all possible values of k. That minimum occurs at $(1+\sqrt{17})/4 \approx 1.281$, and our lower bound for the sum of the two normalized slacks is approximately 1.41.

Corollary 51. *The sum of the normalized →slack and ←slack of the interval filter method, with arbitrary k, is asymptotically (for large h and c) greater than 1.408.*

5. Interval Filter Methods

5.1. Definition of Interval Filter Chunking.

Any local chunking method might be called an interval filter method, because whether a position i is a cutpoint of a file F is determined by applying some criterion (or filter) C to the contents of F in some interval $[i-h, i+h]$ around i. (The notations C and h here are from Definition 21.) But we shall use the phrase "interval filter" in a more restrictive sense.

The first restriction is that whether i is a cutpoint will depend on the contents of the file only in an interval $[i-h, i]$. That is, when reading the file from left to right, one can recognize a cutpoint when one gets to it, without having to read any farther in the file. (This presupposes, for technical reasons, that we know where the file ends; see Remark 53 below.)

A second restriction concerns the particular form of the criterion for cutpoints. We assume that the set PFE of potential file entries has been partitioned into two pieces, U and V, and that the cutpoints of a file F are the positions matching the V in the pattern

$$\underbrace{U\dots U}_{h} V \ .$$

More formally:

Definition 52. The *interval filter chunking* with horizon h determined by a partition of PFE into U and V is the chunking method that declares a position i to be a cutpoint of a file F if and only if

- i is an h-internal position in F.
- all of $F(i-h), \dots, F(i-1)$ are in U, and
- $F(i) \in V$.

This definition immediately shows that interval filter chunking is a local chunking method with horizon h.

Remark 53. For i to be a cutpoint of F, the first clause in the definition requires that the whole interval $[i-h, i+h]$ be included in the domain of F, but the values of F on the right part, $[i+1, i+h]$, of this interval are irrelevant. We could modify the first clause to require only that $[i-h, i]$ be included in the domain of F. Then there could be cutpoints within h of the end of a finite file. The modification would have the advantage that one could determine whether i is a cutpoint of F by reading F up to position i, without needing to know how much farther F goes. The modification would have the disadvantage of violating our definition of locality of chunking methods (Definition 21), which demands that all cutpoints be h internal. Both the advantage and the disadvantage are quite small; in particular, one ordinarily knows where a file ends. It is convenient for our purposes to use the definition as given, ensuring locality.

The particular form of the filter, h consecutive U's followed by a V, ensures that no two distinct cutpoints will be within h positions of each other. Consequently, all chunks have length $> h$ except that the first chunk (in a finite or singly infinite file) might have length only h.

5.2. Statistics of Interval Filter Chunking.

In the following discussion of interval filter chunking, we assume that not only PFE but also U, V, and h are fixed. We use the notations

$$u = \frac{|U|}{|\mathrm{PFE}|} \quad \text{and} \quad v = 1-u = \frac{|V|}{|\mathrm{PFE}|}$$

for the probabilities that a random element of PFE is in U and V, respectively. The independence of the entries at different positions in a random file immediately implies that an arbitrary position is a cutpoint with probability $p = u^h v$. And then, by Proposition 25, the expectation of the chunk length is $\mathbf{E}(L|\mathrm{Cut0}) = 1/(u^h v)$. The following computation will give us the variance of the chunk

length L; it will also give the expectation without the need for Kac's theorem. Since we are concerned here with the chunk length, which was defined only for files with a cutpoint at 0, the probabilities, expectations, etc., in the following discussion are all conditional on the event Cut0.

Let S_k be the probability that the next cutpoint is at k and therefore the chunk length is k. Since the filter prevents cutpoints from being within h of each other, we have $S_k = 0$ for all $k \leq h$. For larger k, we have

$$S_k = u^h v \left(1 - \sum_{j=0}^{k-h-1} S_j\right).$$

The factor $u^h v$ here is the probability that k is a cutpoint. The remaining factor is the probability, conditional on k being a cutpoint, that there is no earlier cutpoint after 0, i.e., that no j in the range $[1, k-1]$ is a cutpoint. The upper limit on the sum is not $k-1$ but $k-h-1$ because the condition that k is a cutpoint already implies that no $j \in [k-h, k-1]$ is a cutpoint. A priori, the terms in the sum should be conditional probabilities of j being a cutpoint, conditional on k being a cutpoint. Fortunately, the conditioning here doesn't matter. Whether j is a cutpoint depends only on file entries at positions $\leq j \leq k-h-1$, while the condition that k is a cutpoint depends only on entries in positions $\geq k-h$. Notice also that the lower limit $j = 0$ of the sum could be replaced by $j = h+1$ or anything in between, as $S_j = 0$ for $j \leq h$; the same observation applies to other sums, like the one defining $S(z)$ below.

To easily manipulate the formula for S_k, we introduce the generating function

$$S(z) := \sum_{k>h} S_k z^k.$$

Multiplying the formula for S_k by z^k and summing over k, we get

$$S(z) = u^h v \sum_{k>h} z^k - u^h v \sum_{k>h} \sum_{j=0}^{k-h-1} S_j z^k.$$

The first term here involves a geometric series and thus simplifies to $u^h v z^{h+1}/(1-z)$. To evaluate the sum in the second term, we interchange the order of summation; since the range of the variables j and k is given simply by $0 \leq j < k-h$, we get

$$\begin{aligned}\sum_{j=0}^{\infty} \sum_{k=h+j+1}^{\infty} S_j z^k &= \sum_{j=0}^{\infty} S_j \frac{z^{h+j+1}}{1-z} \\ &= \frac{z^{h+1}}{1-z} \sum_{j=0}^{\infty} S_j z^j \\ &= \frac{z^{h+1}}{1-z} S(z).\end{aligned}$$

Combining the two terms and remembering that $u^h v$ equals the cutpoint probability p, we get

$$S(z) = \frac{p z^{h+1}}{1-z}(1 - S(z)),$$

and solving for $S(z)$ we get

$$S(z) = \frac{p z^{h+1}}{1 - z + p z^{h+1}}.$$

The expectation of L is the derivative of $S(z)$ evaluated at $z = 1$, so we compute

$$S'(z) = \frac{p z^h (h + 1 - hz)}{(1 - z + p z^{h+1})^2}.$$

When $z = 1$, this reduces to $1/p$, as predicted by Kac's theorem via Proposition 25. We get $\mathbf{E}(L^2)$ by multiplying $S'(z)$ by z, differentiating again, and then setting $z = 1$, because

$$\begin{aligned}(zS'(z))' &= \left(z\sum_k S_k k z^{k-1}\right)' \\ &= \left(\sum_k S_k k z^k\right)' \\ &= \sum_k S_k k^2 z^{k-1},\end{aligned}$$

and setting $z = 1$ yields $\sum_k S_k k^2 = \mathbf{E}(L^2)$. So we differentiate

$$pz^{h+1}(h+1-hz)(1-z+pz^{h+1})^{-2},$$

set $z = 1$ in the result, and simplify to get

$$\mathbf{E}(L^2) = \frac{2}{p^2} - \frac{2h+1}{p}.$$

Finally, the variance of the chunk length is

$$\mathbf{Var}(L) = \mathbf{E}(L^2) - \mathbf{E}(L)^2 = \frac{1}{p^2} - \frac{2h+1}{p} = \frac{1-(2h+1)p}{p^2}.$$

Comparing this to Corollary 43, we see that, for the same cutpoint probability p (hence the same average chunk length $1/p$) and the same minimum chunk length h, our interval filter method gives a smaller variance $\mathbf{Var}(L)$ than the LBFS method. As indicated in the introduction, smaller variance of the chunk length is generally desirable.

As explained earlier, there are other ways to assess desirability of chunking methods. The slack and reverse slack, introduced for just this purpose, and also the probability of long chunks will be considered below. But first, we look briefly at the the criterion that simply compares the expected and minimum chunk lengths.

Having enforced a minimum chunk length of h by our choice of filter, it is reasonable to aim next for chunks that are not too big, and one might do this by choosing the parameters u and v (subject to $u+v = 1$ of course) so as to minimize $\mathbf{E}(L)$. To do this, it is convenient to work with the reciprocal of $\mathbf{E}(L)$, namely $u^h v = u^h(1-u)$, which we want to maximize. Differentiating it with respect to u, we get $hu^{h-1} - (h+1)u^h$, which vanishes at $u = 0$ and at $u = h/(h+1)$. Unless $h = 0$ (in which case we would be dealing with a pure point filter method), $u = 0$ minimizes $u^h v$; the maximum we want is at $u = h/(h+1)$, so $v = 1/(h+1)$. The cutpoint probability for this choice of u and v is

$$u^h v = \frac{h^h}{(h+1)^{h+1}} = \left(1 - \frac{1}{h+1}\right)^h \frac{1}{h+1} \approx \frac{1}{eh},$$

and so the expected chunk length is approximately eh. The average chunk is approximately e times the minimum possible chunk size.

5.3. **Slack of the Interval Filter Method.** Consider a random $F \in \mathrm{PFE}^{\mathbb{M}}$, as in the definition of slack, and use the notation F_1, F_2, F_1^-, F_2^-, F^+ as there. Write P_k for the probability that the first common cutpoint ≥ 0 of the files F_1 and F_2 is at k. So $\mathbf{E}(\text{Slack}) = \sum_k P_k k$.

For $0 \leq k < h$, the probability that k is a common cutpoint is $u^{2h-k}v$. The reason is that for k to be a common cutpoint requires $F^+(k)$ to be in V (probability v), $F^+(j)$ to be in U for $j = 0, 1, \dots, k-1$ (k events of probability u each), and both $F_1^-(j)$ and $F_2^-(j)$ to be in U for $j = k-h, \dots, -1$ ($2(h-k)$ events of probability u each). All these events are independent, so we just multiply their probabilities. Furthermore, still assuming $0 \leq k < h$, we know that, if k is a common cutpoint, then it is the first one, because distinct cutpoints can never be within h of each other. Thus,

$$P_k = u^{2h-k}v \qquad \text{for } 0 \leq k < h.$$

For $k \geq h$, on the other hand, the probability that k is a common cutpoint is given by the simpler formula $u^h v$, since we just require $F(k)$ to be in V and $F(j)$ to be in U for the h values $k-h, \dots, k-1$ of j. But the probability that k is the first common cutpoint is more complicated,

since we must exclude the possibility of earlier common cutpoints. More precisely, when k is a common cutpoint, then none of $k-h,\dots,k-1$ can be a cutpoint (of either file), but we must exclude the possibility of a common cutpoint $j \in [0, k-h-1]$. Note that, since we only consider values of j smaller than $k-h$, the event that such a j is a cutpoint is independent of the event that k is a cutpoint; indeed, the former depends on file contents at positions $\le j < k-h$ while the latter depends on file contents at positions $\ge k-h$, and these are independent. Thus, we have

$$P_k = u^h v \left(1 - \sum_{j=0}^{k-h-1} P_j\right) \qquad \text{for } k \ge h.$$

Note that, although the events "j is a common cutpoint" for distinct values of j need not be mutually exclusive (if the j values differ by more than h), the events "j is the first common cutpoint" are mutually exclusive, so the sum in our formula correctly represents the probability of their union — and this is precisely the probability that some $j < k-h$ is a common cutpoint.

As before, manipulation of these formulas for P_k becomes easier if we introduce the generating function

$$P(z) = \sum_{k=0}^{\infty} P_k z^k.$$

If we take the formulas above for P_k, multiply by z^k and sum over k, we get

$$P(z) = \sum_{k=0}^{h-1} u^{2h-k} v z^k + \sum_{k=h}^{\infty} u^h v z^k - \sum_{k=h}^{\infty} u^h v \sum_{j=0}^{k-h-1} P_j z^k.$$

We simplify each of the three terms on the right, and for notational convenience we remember that the product $u^h v$ is the cutpoint probability, for which we have the shorter notation p. The first term on the right side is

$$u^{2h} v \sum_{k=0}^{h-1} \left(\frac{z}{u}\right)^k = pu^h \frac{\left(\frac{z}{u}\right)^h - 1}{\frac{z}{u} - 1}.$$

We introduce the abbreviation

$$\alpha(z) = \frac{\left(\frac{z}{u}\right)^h - 1}{\frac{z}{u} - 1},$$

so that the first term on the right side of the formula for $P(z)$ becomes $pu^h\alpha(z)$. The second term is

$$\sum_{k=h}^{\infty} pz^k = \frac{pz^h}{1-z}.$$

In the third term, we interchange the order of summation, obtaining

$$u^h v \sum_{j=0}^{\infty} \sum_{k=j+h+1}^{\infty} P_j z^k = p \sum_{j=0}^{\infty} P_j \frac{z^{h+j+1}}{1-z} = \frac{pz^{h+1}}{1-z} P(z).$$

Inserting these simplifications into our formula for $P(z)$ and multiplying by $1-z$ to clear denominators, we get

$$(1-z)P(z) = (1-z)pu^h\alpha(z) + pz^h - pz^{h+1}P(z).$$

The expectation of the slack is obtained by differentiating $P(z)$ to get $\sum_k P_k k z^{k-1}$ and then setting $z = 1$ to get $\sum_k P_k k$. To evaluate $P'(1)$, we differentiate the last displayed formula with respect to z and substitute $z = 1$ in the result. The differentiation is simplified by the observation that, whenever a factor $1-z$ in our formula survives in the derivative, it will be annihilated by the substitution; in particular, we can ignore the term involving $\alpha'(z)$. The computation produces

$$-P(1) = -pu^h\alpha(1) + hp - (h+1)pP(1) - pP'(1).$$

Remembering that $P(1) = \sum_k P_k = 1$ and solving for $P'(1)$, we get

$$P'(1) = \frac{1}{p} - u^h\alpha(1) - 1.$$

The definition of $\alpha(z)$ gives

$$u^h \alpha(1) = u^h \frac{\left(\frac{1}{u}\right)^h - 1}{\frac{1}{u} - 1} = \frac{1 - u^h}{\frac{1}{u} - 1}.$$

Inserting this into the formula for $P'(1)$, we finally get, using also that $1 - u = v$,

$$\mathbf{E}(\text{Slack}) = \frac{1}{p} - \frac{\frac{1}{u} - u^h}{\frac{1}{u} - 1} = \frac{1}{p} - \frac{1 - u^{h+1}}{v}$$

The expectation of the normalized slack is obtained by dividing by the expected chunk length, i.e., multiplying by $p = u^h v$, which gives

$$1 - u^h + u^{2h+1}.$$

If, as suggested at the end of the preceding subsection, we choose $u = h/(h+1)$, so as to minimize the expected chunk length, then the normalized slack is, on average

$$1 - \left(1 - \frac{1}{h+1}\right)^h + \left(1 - \frac{1}{h+1}\right)^{2h+1} \approx 1 - \frac{1}{e} + \frac{1}{e^2},$$

where the approximation is good for large h.

On the other hand, one might want to choose u and v so as to minimize the expected normalized slack. By differentiating, one finds that the minimum occurs when $u^h = (h+1)/(2h+1) \approx 1/2$,

$$u \approx \sqrt[h]{\frac{1}{2}} \quad \text{and so} \quad v \approx \frac{\ln 2}{h}.$$

The minimum value of the expected normalized slack is thus approximately 3/4.

5.4. **Probability of Long Chunks.** We estimate next the probability of getting long chunks in the interval filter method. This estimate will be relevant later in two computations. One concerns the probability of a long interval containing no cutpoints; the other concerns the right slack of the method.

As before, we assume that u, v, and h are given, and we write p for the cutpoint probability $p = u^h v$.

Let q_k be the conditional probability that there is no cutpoint in the interval $[1, k]$ given that there is a cutpoint at 0. Let $Q(z)$ be the generating function

$$Q(z) = \sum_{k=0}^{\infty} q_k z^k.$$

Before proceeding to estimate q_k, we comment on the condition "there is a cutpoint at 0." It obviously implies that the file entry at 0 is in V. It implies more, namely that the h previous file entries are in U, but this additional information has no effect on the probability of a cutpoint at any positive position. To get a cutpoint at some $k > 0$, we need h consecutive elements of U at positions $k - h$ to $k - 1$ and, because of the element of V at position 0, no positions farther to the left can contribute to a cutpoint at k. Thus, q_k could also be described as the probability that a random file F has a cutpoint at k, given that $F(0) \in V$.

We have almost computed the probabilities q_k in Section 5.2. We obtained there the generating function $S(z)$ for the probabilities S_k that the first positive cutpoint is at k, given a cutpoint at 0. This S_k can be described as the probability, given a cutpoint at 0, that there is no cutpoint in $[1, k-1]$ but there is one in $[1, k]$. That is,

$$S_k = q_{k-1} - q_k$$

for all $k \geq 1$. Multiplying this equation by z^k, summing over all $k \geq 1$, and taking into account that $S_0 = 0$ and $q_0 = 1$, we find that

$$S(z) = zQ(z) - Q(z) + 1.$$

Solving for $Q(z)$ and using the formula for $S(z)$ computed in Section 5.2, we find that

$$Q(z) = \frac{1 - S(z)}{1 - z} = \frac{1}{1 - z + pz^{h+1}}.$$

Because this generating function is rational, one can, in principle, expand it in partial fractions, expand those fractions as geometric series, and thus obtain explicit formulas for the probabilities q_k. This approach, unfortunately, presupposes that one knows the roots of the polynomial $1 - z + pz^{h+1}$; these roots enter into the partial fraction expansion, and (unless there are multiple roots) the formula for q_k is a linear combination (with constant coefficients) of the $h+1$ terms w^k, where w ranges over the reciprocals of the $h+1$ roots of the polynomial. In other words, w ranges over the solutions of the equation $w^{h+1} - w^h + p = 0$, which is most conveniently (for our purposes) written as $w^h(1-w) = p$. For large k, the formula for q_k will be dominated by the w^k term for the largest of the roots. Most of the rest of this section will be devoted to estimating this largest root and thus estimating the rate at which q_k approaches 0 as $k \to \infty$. But first we indicate, in the following remark, an alternative approach to the computation of q_k; the reader can safely skip it, as it will not be used directly in the subsequent work.

Remark 54. The inclusion-exclusion principle provides a formula for q_k as follows. For each subset A of $\{1, 2, \dots, k\}$, let $r(A)$ be the probability that a random file with a cutpoint at 0 also has cutpoints at all the members of A (and possibly additional points as well). Notice that $r(A) = 0$ if any element of A is $\leq h$ or if any two distinct elements differ by $\leq h$, because no file has two cutpoints separated by a distance h or less. For all other choices of A, the events of having cutpoints at the various elements of A are independent of each other and of the condition that there is a cutpoint at 0, so $r(A) = p^{|A|}$. The inclusion-exclusion principle, applied to this situation, says that

$$q_k = \sum_{A \subseteq \{1,\dots,k\}} (-1)^{|A|} r(A) = \sum_l (-1)^l p^l N_l,$$

where N_l is the number of l-element subsets of $\{1, 2, \dots, k\}$ that are good in the sense that they have no elements $\leq h$ and no two distinct elements a distance $\leq h$ apart. Fortunately, these good subsets are quite easy to count. Notice first that, if $A = \{a_1 < a_2 < \cdots < a_l\}$ is good, then $0 < a_1 - h$ (because A has no elements $\leq h$) and $a_1 - h < a_2 - 2h < \cdots < a_l - lh$ (because a_i and a_{i+1} differ by more than h). Thus, we can compress A to a set of l positive integers $\tilde{A} = \{a_i - ih : 1 \leq i \leq l\}$, which is obviously a subset of $\{1, 2, \dots, k - lh\}$. Conversely, any l-element subset of $\{1, 2, \dots, k - lh\}$ arises as $\tilde{A}$ from a unique good A. Therefore,

$$N_l = \binom{k - lh}{l}$$

and

$$q_k = \sum_l (-1)^l p^l \binom{k - lh}{l}.$$

(The variable l can be allowed to range over all integers, but the binomial coefficient will vanish unless l is in the range of reasonable values, $0 \leq l \leq k/(h+1)$.) This formula for q_k, though quite explicit, does not seem to be directly amenable to estimating the asymptotic behavior of the sequence of q_k's, mainly because of cancellations between the positive and negative terms in the sum. It leads, however, to a recursive formula that can be more useful for asymptotic estimates. Indeed, the familiar Pascal-triangle identity for the binomial coefficients allows us to write

$$q_k = \sum_l (-1)^l p^l \binom{k - lh - 1}{l} + \sum_l (-1)^l p^l \binom{k - lh - 1}{l - 1}.$$

The first sum here is simply q_{k-1}. The second can be rewritten, by factoring out $-p$ and changing the summation variable to $m = l - 1$, as

$$-p \sum_l (-1)^m p^m \binom{k - h - 1 - mh}{m} = -p q_{k-h-1}.$$

Therefore, we get the linear recursion equation

$$q_k = q_{k-1} - p q_{k-h-1}.$$

The usual technique for solving such recursions, namely looking for constants λ such that λ^k solves the recursion, forming a linear combination of such solutions for the various possible λ's,

and choosing the constant coefficients in the linear combination to match initial conditions, leads to the following equation for λ:

$$\lambda^{h+1} = \lambda^h - p.$$

This is exactly the equation whose roots w figured in the discussion preceding this remark. So we have reached, via a different route, the same conclusion: The asymptotic behavior of q_k for large k is exponential, $q_k \approx \text{constant} \cdot \lambda^k$, where λ is the largest (in absolute value) root of $w^h(1-w) = p$.

We now turn to the task of estimating the largest root of $w^h(1-w) = p$. Recall that the cutpoint probability p was obtained as $p = u^h v = u^h(1-u)$, so u is a root of our equation. Recall also that, for a fixed h, the maximum possible value of p is $p_{\max} = h^h/(h+1)^{h+1}$, attained at $u = h/(h+1)$.

To study the solutions of $w^h(1-w) = p$, let us concentrate first on non-negative real solutions, and so let us consider the graph of the function $f(w) = w^h(1-w)$. For non-negative w, it starts at the origin, where it has a root of multiplicity h; it increases up to the point where $w = h/(h+1)$ and $f(w) = h^h/(h+1)^{h+1}$ (the maximum we computed earlier), and then it decreases to the point where $w = 1$ and $f(w) = 0$. As w continues to increase, $f(w)$ continues to decrease, i.e., to become more and more negative.

Thus, for p in the range $0 \le p < p_{\max}$, there are two non-negative real solutions w for $w^h(1-w) = p$, both of which are simple roots except that when $p = 0$ the root $w = 0$ has multiplicity h. Let us write w_+ for the larger and w_- for the smaller of these roots. For $p = p_{\max}$, there is a single non-negative real root, with multiplicity 2; we let both w_+ and w_- denote this root. Thus, both w_+ and w_- are continuous functions of p in $[0, p_{\max}]$. We can ignore any values of p outside this interval, because they cannot arise as the cutpoint probability of the interval filter method.

We propose now to show that w_+ is the root of largest absolute value for the equation $w^h(1-w) = p$, not just among the non-negative real roots already considered but also among all roots in the complex plane. Notice that there is no problem (and no interest) in the case $p = 0$; here we know all the roots: an h-fold root at $0 = w_-$ and a simple root at $1 = w_+$. So we may assume $p \in (0, p_{\max}]$. Temporarily fix p (and thus w_+) and consider the following two disks in the complex plane. The *left disk* is centered at the origin and has radius w_+; the *right disk* is centered at 1 and has radius $1 - w_+$. The situation is illustrated in Figure 1. Notice that the two disks are tangent at their common boundary point w_+. Our ultimate goal is to show that all the roots of $w^h(1-w) = p$ lie in the left disk, but first we establish the easier result that all these roots lie in the union of the two disks.

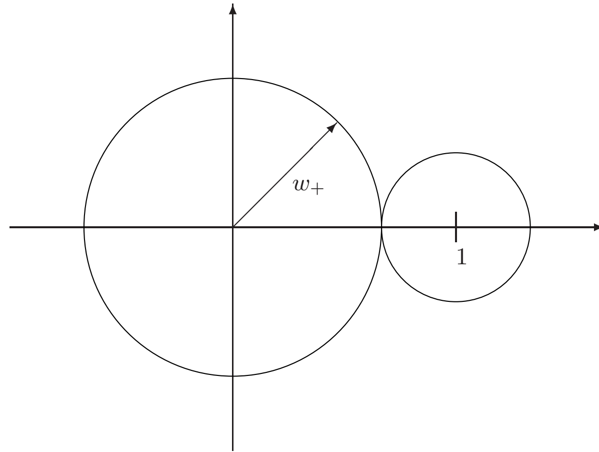


FIGURE 1. A cover for the roots of $w^h(1-w) = p$

To see this, suppose w is a root that lies outside the left disk. So $|w| > w_+$ and thus $|w|^h > {w_+}^h$. But

$$|w|^h \cdot |1-w| = p = {w_+}^h \cdot (1 - w_+),$$

and so we must have $|1 - w| < 1 - w_+$. That is, w lies in the right disk.

To show that the roots actually all lie in the left disk, we allow p to vary and consider what happens if p starts at 0 and gradually increases toward $p_{\max}$. Of course, at each stage of the process, there is a root w_+ at the point of tangency of our two disks. Our concern is with the behavior of the other h roots. Initially, when $p = 0$, these roots are all at the center 0 of the left disk. (The left disk is initially the unit disk and the right disk a single point.) Now as p increases, the left disk shrinks, the right disk grows, and the roots we are interested in move around continuously. Can they escape from the left disk? The preceding paragraph shows that the only way to escape from the left disk is to enter the right disk. Since the disks touch only at w_+, an escaping root would have to coincide, at the moment of its escape, with w_+. That is, w_+ would have to be (at least) a double root of our equation $w^h(1-w) = p$. But we know, from our analysis of the non-negative real roots, that w_+ is a simple root for $p < p_{\max}$; it becomes a double root only when p reaches $p_{\max}$. Thus, the only possible moment when a root can escape from the left disk is at the very end of the range of relevant p values. That is, for all $p \in [0, p_{\max}]$, all the roots are still in the left disk.

This completes the verification of our claim that w_+ is always the largest root in absolute value.

There remains the question of evaluating or at least estimating w_+ as a function of p (where we continue to regard h as fixed). The two following rough estimates will suffice for our purposes; both are based on the fact that the function $f(w) = w^h(1-w)$ has a negative second derivative throughout the interval $[h/(h+1), 1]$ (in fact for all $w > (h-1)/(h+1)$). Recall that this is the interval over which w_+ ranges as $p = f(w)$ varies from $p_{\max}$ down to 0. Knowing that the graph of f is concave, we have that this graph lies below its tangents and above its chords on this interval.

The tangent to the graph of f at the point $w = 1$, $p = 0$ is the line $p + w = 1$, because $f'(1) = -1$. Since the graph is below this tangent, we conclude that $w_+ \le 1 - p$ (with equality only at $p = 0$).

The chord of the graph of f joining the point $w = 1$, $p = 0$ and the point $w = h/(h+1)$, $p = p_{\max}$ has the equation $w = 1 - p/(p_{\max}(h+1))$. Since the graph lies above the chord, we have

$$w_+ \ge 1 - \frac{1}{p_{\max}(h+1)} p \approx 1 - ep,$$

where the approximation is good for large h. Figure 2 illustrates the above estimations. (For the sake of visibility, the figure is drawn with different scales along the two axes.)

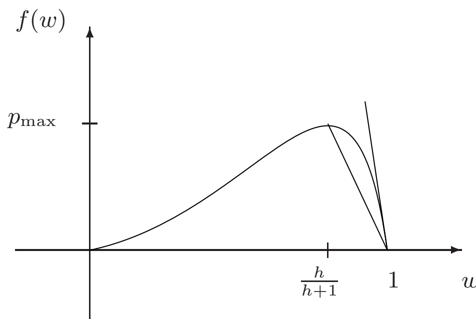


FIGURE 2. A graph of $f(w)$ for estimating w_+ as a function of p

Inserting these estimates of w_+ into our previous results concerning the probabilities q_k of chunks longer than k, we find that, for large k, q_k lies between a constant multiple of $(1-ep)^k$ and a constant multiple of $(1-p)^k$. The constant here arises from the partial fraction expansion of the generating function $Q(z)$. If we express k in terms of the average chunk length $1/p$, i.e., if we set $k = M/p$, so that q_k is the probability that the chunk containing 0 is at least M times the average length, then we can approximate

$$(1-ep)^k = \left((1-ep)^{1/ep}\right)^{eM} \approx e^{-eM}$$

and

$$(1-p)^k = \left((1-p)^{1/p}\right)^M \approx e^{-M}.$$

Thus, as a function of M, the probability of a chunk M times as long as the average decreases exponentially and lies between a constant multiple of e^{-eM} and a constant multiple of e^{-M}. The actual factor in the exponent depends on p; the upper estimate e^{-M} is more accurate for very small p (where the graph of f is close to the tangent used in obtaining this estimate), while the lower estimate e^{-eM} is more accurate for relatively large p, i.e., close to $p_{\max}$.

One can, of course, obtain more accurate estimates of the decay rate of q_k by estimating w_+ more accurately. For example, one can express w_+ as a Taylor series in $1-p$ or as a Puiseux series in $p-p_{\max}$.

5.5. **Reverse Slack of the Interval Filter Method.** The estimates of q_k in Section 5.4 enable us to estimate the reverse slack of the interval filter method as follows. Consider a random $F \in \mathrm{PFE}^{\mathbb{D}}$. The probability that its ←slack is a particular natural number k is the product of the probability of a cutpoint at $-k$ and the conditional probability, given a cutpoint at $-k$, of having no further cutpoints in $[-k+1,0]$. The former factor is $p = u^h v$ and the latter is, thanks to shift-invariance, q_k. (We are in the pleasant situation that neither factor is influenced by F_1^+ and F_2^+; only F^- is relevant, and so we are essentially dealing with a file in $\mathrm{PFE}^{\mathbb{Z}}$.)

The expectation of the ←slack is therefore

$$\mathbf{E}(\leftarrow \text{Slack}) = \sum_{k=0}^{\infty} kpq_k = pQ'(1),$$

where Q is the generating function obtained in Section 5.4,

$$Q(z) = \sum_{k=0}^{\infty} q_k z^k = \frac{1}{1-z+pz^{h+1}}.$$

Differentiating, we find

$$Q'(z) = -(1-z+pz^{h+1})^{-2}(-1+p(h+1)z^h)$$

and so

$$Q'(1) = p^{-2}(1-p(h+1))$$

and

$$\mathbf{E}(\leftarrow \text{Slack}) = \frac{1}{p} - (h+1).$$

The normalized ←slack therefore has expectation $1-p(h+1)$.

If h is fixed, then the value of p that minimizes the normalized ←slack is $p_{\max} = h^h/(h+1)^{h+1}$, and the minimum of the normalized ←slack is

$$1 - p_{\max}(h+1) = 1 - \left(\frac{h}{h+1}\right)^h \approx 1 - \frac{1}{e}$$

for large h.

6. Local Maximum Chunking

As the name suggests, local maximum chunking selects, as the cutpoints of a file F, those positions i where the entry $F(i)$ attains a local maximum. Recall that we fixed a linear ordering of PFE, so the notion of (strict) maximum makes sense. "Local" means that $F(i)$ is a maximum within a radius of h positions to either side. Here is the formal definition.

Definition 55. *Local maximum chunking* with horizon h is the chunking method that declares a position i to be a cutpoint of a file F if and only if

- i is h-internal to F and
- $F(i) > F(j)$ for all $j \neq i$ in the interval $[i-h, i+h]$.

We refer to the interval $[i-h, i+h]$ as the *window* around i. Clearly (and fortunately for the terminology), local maximum chunking is a local chunking method.

We shall often assume that the number $|\mathrm{PFE}|$ of potential file entries is much larger than the horizon h. In applications, we might typically have $|\mathrm{PFE}| = 2^{32}$ and $h = 2^7$, so the assumption is justified in practice. The main purpose of the assumption is to excuse us from paying attention to the strictness of the inequality $F(i) > F(j)$ in the definition. Of course, if we had written $F(i) \geq F(j)$ instead, then there could be more cutpoints in a file, points that are tied for maximum with some other point in their window. When $|\mathrm{PFE}| \gg h$, ties for maximum become very unlikely, so we can safely ignore them when estimating the statistical properties of local maximum chunking. We shall use phrases like "ignoring ties" to refer to the assumption that $|\mathrm{PFE}| \gg h$ and to indicate how it is being used.

Remark 56. According to the so-called "birthday paradox", one would need the stronger inequality $|\mathrm{PFE}| \gg h^2$ to ensure that all ties are unlikely. This is no problem, for two reasons. First, the typical values quoted above satisfy the stronger inequality. Second, we often don't care about avoiding all ties but only ties for maximum, and for this $|\mathrm{PFE}| \gg h$ suffices.

6.1. **Statistics of Local Maximum Chunking.** It is clear from the definition of local maximum chunking that, if i is a cutpoint, then no other point in its window can be a cutpoint. Therefore, all chunks have length at least $h+1$, except that the first chunk (in a finite or singly infinite file) can have length h.

Ignoring ties, we easily see that the cutpoint probability for local maximum chunking is $p = 1/(2h+1)$. Indeed, if i is any h-internal position of a file, then exactly one of the $2h+1$ positions in the window around i must have the largest file entry in this window (because there are no ties for maximum), and each of the positions has an equal chance, $1/(2h+1)$.

It follows, by Kac's theorem via Proposition 25, that the expectation of the chunk length is $2h+1$.

Remark 57. Abstaining from the assumption that there is no tie for the maximum, we still have a formula for the cutpoint probability, namely

$$p = \sum_{k=0}^{m-1} \frac{1}{m}\left(\frac{k}{m}\right)^{2h},$$

where we have abbreviated $|\mathrm{PFE}|$ as m, so $\mathrm{PFE} = \{0, 1, \dots, m-1\}$. The term indexed by k in this sum represents the probability that the file value at i is k (probability $1/m$) and the other $2h$ file values in the window are in the range $[0, k-1]$ (probability k/m for each position and thus $(k/m)^{2h}$ for all $2h$ positions). As we saw in Section 2.3, this sum is approximately $1/(2h+1)$ for large m, in agreement with the "no ties" estimate above. Also, we exhibited in Section 2.3 the more precise formula

$$p = \frac{1}{m^{2h+1}} \frac{1}{2h+1} \sum_{k=0}^{2h} \binom{2h+1}{k} B_k m^{2h+1-k} = \frac{1}{2h+1} - \frac{1}{2m} + \frac{h}{6m^2} + \dots .$$

As $m \to \infty$, all terms except $1/(2h+1)$ approach zero.

Unfortunately, we do not have good estimates for the slack of the local maximum chunking method or for the variance of the chunk length. We briefly indicate in this section what we know and where the difficulty arises in trying to go farther.

Consider first the slack. Because of the left-right symmetry of the local maximum method, it is clear that the expectations of the slack and reverse slack are equal. We therefore confine attention to the former.

Consider a random $F \in \mathrm{PFE}^{\mathbb{M}}$. What is the probability that its slack is a particular number $k \geq 0$? For $k \leq h$, all that is required is that k be a common cutpoint of F_1 and F_2; it will be the first non-negative cutpoint because cutpoints are never within a distance h of each other. The windows centered at k in the two files together contain $3h-k+1$ positions, namely the $k+h+1$ non-negative positions from 0 through $k+h$ and $2(h-k)$ negative positions, $k-h$ through -1 in each of the two files. Thus, the probability that k is the slack is $1/(3h-k+1)$.

Consequently, the probability that the slack is at most h is

$$\sum_{k=0}^{h} \frac{1}{3h-k+1} = \sum_{j=2h+1}^{3h+1} \frac{1}{j} \approx \ln(3h+1) - \ln(2h) \approx \ln\frac{3}{2}.$$

For $k > h$, however, the situation is more complicated. The probability that k is a common cutpoint is actually simpler; it is $1/(2h+1)$ because the window $[k-h, k+h]$ lies entirely in the common part F^+ of the two files. What is difficult is finding the probability, given that k is a cutpoint, that it is the first one ≥ 0. The fact that k is a cutpoint gives some information about the values of F^+ in $[k-h, k-1]$; they are more likely to be smaller than they would be if we knew nothing about k, for all these values are $\leq F(k)$. And that increases the probability that positions just to the left of this window (positions $k-2h$ to $k-h+1$) are cutpoints; their windows overlap the region where F^+ has smaller than usual values, so they have a better chance of being a local maximum. It is this indirect effect of the condition "k is a cutpoint" on the cutpoint probabilities between h and $2h$ positions earlier that is difficult to compute and has prevented us from estimating the slack of the local maximum method.

The problem of computing the slack is indirectly related to the problem of computing the variance of the chunk length, as follows. Obeying Pólya's dictum (as quoted by Halmos in [14]), "if you can't solve a problem, then there is an easier problem that you can't solve — find it!" notice that the difficulty described above, which prevents us from computing the expected slack of a random $F \in \mathrm{PFE}^{\mathbb{M}}$, also affects the simpler case of a random file $F \in \mathrm{PFE}^{\mathbb{Z}}$, if we consider the obvious analog of the slack, namely the smallest non-negative cutpoint. We know that each point $k \in \mathbb{Z}$ has probability $1/(2h+1)$ of being a cutpoint, but its probability of being the first non-negative cutpoint is subject, when $k > h$, to the same complications encountered in trying to compute the slack.

Instead of the first non-negative cutpoint of a doubly infinite file F, it is convenient to consider the first positive cutpoint, which we call $\rho(F)$. The expectation is merely increased by 1, because the chunking method is shift-invariant. The notation ρ matches that of Section 2.2, the number of iterations of the Bernoulli shift needed to bring F into the set Cut0 of files with a cutpoint at 0. So the expectation of the first positive cutpoint is given by Proposition 11 in terms of the probability of Cut0 and the conditional variance of ρ, conditional on the event Cut0. The probability of Cut0 is just the cutpoint probability, $1/(2h+1)$. And $\mathbf{Var}(\rho|A)$ is just the variance of the chunk length.

Thus, the problem of computing the variance of the chunk length turns out to be equivalent to a simplified version of the problem of computing the average slack. Note that this equivalence is not specific to local maximum chunking but applies to any shift-invariant chunking method for which the average chunk length is known.

7. Probability of Long Chunks

This section is about another measure of quality of local chunking methods, namely the probability of getting exceptionally large chunks. More precisely, we deal with the probability that a long interval $[1, l]$ contains no cutpoints of a random, doubly infinite file. Since very large chunks are undesirable, one wants this probability to be small. But one does not want to achieve this by making all the chunks too small, since very small chunks are also undesirable. In order to fairly compare different chunking methods, it is therefore reasonable to choose their parameters so that the average chunk sizes agree and then to ask about the probabilities of significantly larger chunks. In this section, we carry out a comparison of the chunking methods we have discussed, computing the probability of finding no cutpoint in an interval $[1, l]$ where l is a specified multiple of the average chunk size.

As we want to show that the probability of such a long cut-less interval is smaller (and thus better) for the local maximum method than for its competitors, we shall estimate the former from above and the latter from below.

The most complicated computation here will be for the local maximum method, so we arrange our notation and conventions to maximize convenience there. As before, we let h be the horizon for the local maximum method, so the expected chunk length is $2h + 1$. We shall estimate the probability of finding no cutpoints in the interval $[1, 2hM]$, which is essentially (glossing over the

distinction between $2h$ and $2h+1$) M average chunk lengths long. Here the multiplier M should be larger than 1, but it need not be huge. $M = 5$ might be a reasonable choice; i.e., we might want to have low probability that a particular chunk is more than 5 times the average length.

Our calculation will assume that there are no ties among the file entries in the interval under consideration. Considerably weaker assumptions would suffice, but this one is easy to use and is satisfied with high probability for typical values of the parameters.

7.1. **Pure Point Filter Method.** We first obtain a lower estimate for the probability of long chunks in the pure point filter method. As indicated above, for a fair comparison, we adjust the parameter c of the pure point filter method to produce the same average chunk length $2h+1$ as the local maximum method. Then each position in a random file has, independently, probability $1/(2h+1)$ of being a cutpoint. Therefore, the probability of no cutpoint among $2hM$ points is

$$\left(1 - \frac{1}{2h+1}\right)^{2hM}.$$

We resist the temptation to approximate $1 - (1/(2h+1))$ by $e^{-1/(2h+1)}$, because this is an approximation from above and we want one from below. Instead, we use that

$$\frac{2h+1}{2h} = 1 + \frac{1}{2h} < e^{1/(2h)},$$

so

$$1 - \frac{1}{2h+1} = \frac{2h}{2h+1} > e^{-1/(2h)},$$

and therefore the probability of no cut point in $[1, 2hM]$ in the pure point filter model is

$$\left(1 - \frac{1}{2h+1}\right)^{2hM} > e^{-2hM/2h} = e^{-M}.$$

7.2. **LBFS Method.** We next perform the analogous computation for the LBFS chunking method. That method had two parameters, previously called c and h. The latter notation is no longer usable, since we are now using h as the horizon of the local maximum method. Fortunately, in our earlier discussion of the LBFS method, we introduced the notation k for the ratio h/c, and this letter is still available. So we shall carry out the computation for the LBFS method with $1/c$ as the probability of any position being a candidate and with kc as the horizon. We think of k as fixed; for example k was $1/4$ in the version of the LBFS method proposed in [20]. We adjust c to make the expected chunk length $c+kc$ match our $2h+1$. Thus, $c = (2h+1)/(k+1)$.

Now the probability that the interval $[1, 2hM]$ contains no cutpoint is obviously bounded below by the probability that this interval contains no candidate, namely

$$\left(1 - \frac{1}{c}\right)^{2hM} = \left(1 - \frac{k+1}{2h+1}\right)^{2hM}.$$

We estimate this from below by the same technique already used for the pure point filter method. We have

$$\frac{2h+1}{2h-k} = 1 + \frac{k+1}{2h-k} \le e^{\frac{k+1}{2h-k}}$$

and therefore

$$1 - \frac{k+1}{2h+1} = \frac{2h-k}{2h+1} \ge e^{-\frac{k+1}{2h-k}}.$$

Thus, the probability that there is no cutpoint in $[1, 2hM]$ is at least

$$e^{-(k+1)2hM/(2h-k)}.$$

This estimate is useless for very large k, but for reasonable k, small compared to h, this lower bound is essentially $e^{-(k+1)M}$. For example, when $k = 1/4$, we have the lower bound $e^{-5M/4}$. More generally, as long as $k \ll h$, we get a lower bound that decreases only exponentially with M.

7.3. **Interval Filter Method.** Although our main goal is to show that the local maximum method makes long intervals without cutpoints far less likely than the previously known LBFS chunking method, we include also a rough estimate for the corresponding probability in the interval filter method. As with the point filter and LBFS chunking methods, we shall find that the probability of finding no cutpoint in a long interval decreases (only) exponentially with respect to the interval's length.

We already computed in Section 5.4 the conditional probability q_l of finding no cutpoint in $[1, l]$ given that there is a cutpoint at 0. Now we shall compute the unconditional probability of finding no cutpoint in $[1, l]$. We shall obtain it by combining conditional probabilities, the conditions being the various possibilities for what happens at and shortly before position 0. Since we are interested in long intervals, we shall assume that $l > h$.

For each j in the range $0 \leq j < h$, let C_j be the event that the entry $F(-j)$ at position $-j$ is in V and all the entries in positions $-j+1$ to 0 are in U. So $\mathbf{Prob}(C_j) = vu^j$. The conditional probability, given C_j, of having no cutpoint in $[1, l]$ is

$$u^l + \sum_{i=1}^{h-j} u^{i-1} v q_{l-i}.$$

The first term here is the probability that there is no element of V, and therefore certainly no cutpoint, at any position in $[1, l]$. The term indexed by i in the sum is the probability that V occurs somewhere in the interval $[1, l]$, that the first such occurrence is at i, and that no cutpoint occurs thereafter, in $[i+1, l]$. We restrict i to range only up to $h - j$ because, if the first positive i where V occurs were at position $h - j + 1$ or later, then it would be preceded by at least h consecutive U's, from position $-j+1$ to $i-1$ inclusive. Then this i would be a cutpoint, so this situation does not contribute to the probability we are computing.

Let D be the event that none of the C_j occur, i.e., the event that all file entries from position $-h+1$ to 0 are U's. This event has probability u^h and the conditional probability, given D, of finding no cutpoint in $[1, l]$ is simply the probability that all l of the file entries from position 1 to l are in U; the reason is that, if there were any V in this range, then the first one would, because of D, be a cutpoint.

Thus, the probability of finding no cutpoint in $[1, l]$ is given by

$$\sum_{j=0}^{h-1} u^j v \left(u^l + \sum_{i=1}^{h-j} u^{i-1} v q_{l-i} \right) + u^{h+l}.$$

The sum over j of $u^j v u^l$ is just a finite geometric series; evaluating it and remembering that $1 - u = v$, we find simply $u^l - u^{h+l}$. The second term here cancels the u^{h+l} that arose from D, so what remains is u^l. The double sum over j and i can be simplified somewhat by reversing the order of summation. The final result is that the probability that the interval $[1, l]$ contains no cutpoint is

$$u^l + v \sum_{i=1}^{h} q_{l-i} u^{i-1} - p \sum_{i=1}^{h} q_{l-i}.$$

Note that i ranges only up to h, so for large (compared to h) values of l, all the subscripts of q's in this formula are large, so we can use the asymptotic estimates from Section 5.4. Thus, we find that all terms in our formula decrease exponentially as l grows; specifically the probability of finding no cutpoint in $[1, l]$ is asymptotically a constant multiple of ${w_+}^l$. From our rough estimates of w_+ in Section 5.4, we can infer that, if $l = M/p$, so that the interval $[1, l]$ is M times as long as an average chunk, then the probability that $[1, l]$ contains no cutpoint is asymptotically Ae^{-BM} for some constants A and B, with B lying between 1 and e.

7.4. **Local Maximum Method.** In this subsection, we estimate from above the probability that the local maximum method with horizon h produces no cutpoint in the interval $[1, 2hM]$. Recall that we assume that $|\mathrm{PFE}|$ is so large that we can ignore the possibility of two relevant positions having equal entries in a random file. Here the relevant positions are not just the interval $[1, 2hM]$ but an additional h positions at either end, since these are within the windows of positions 1 and $2hM$ and may thus affect cutpoints within $[1, 2hM]$.

Observe that whether a position i is a local maximum depends only on the relative ordering of the values of F on the interval $[i-h, i+h]$, not on the actual values of F. Thus, we could replace our probability space of random files with the finite space of all linear orderings of the relevant interval of positions $[1-h, 2hM+h]$; all of the $(L+2h)!$ orderings are equally probable.

Formally, this means that we use the function Φ that assigns to each $F \in \mathrm{PFE}^{\mathbb{Z}}$ (without ties in the relevant segment) the ordering induced on $[1-h, 2hM+h]$ by

$$m \prec n \iff F(m) < F(n),$$

we observe that our probability measure on $\mathrm{PFE}^{\mathbb{Z}}$ projects via Φ to the uniform measure on the set of orderings, and we observe that the notion of local maximum (as well as the other notions that will play a role in our computations) depend on F only via $\Phi(F)$.

7.4.1. *Splitting intervals without local maxima.* To estimate from above the probability of the event "no local maximum in $[1, 2hM]$", we first show that this event is included in some other events whose probability is easier to estimate. That is, we analyze sequences F that have no local maximum in $[1, 2hM]$, and we establish some other properties that all such sequences must have.

Accordingly, we consider a temporarily fixed F with no local maximum in $[1, 2hM]$, and we deduce some properties of F.

For each $n \in [1, 2hM]$, define $\mu(n)$ to be the element of $[n-h, n+h]$ where F has the largest value. Since n is not a local maximum, $\mu(n) \neq n$. Partition $[1, 2hM]$ into two pieces according to the relative order of $\mu(n)$ and n; that is, define

$$A = \{n \in [1, 2hM] : \mu(n) < n\} \quad \text{and} \quad B = \{n \in [1, 2hM] : \mu(n) > n\}.$$

Lemma 58. *A is an initial segment and B a final segment of* $[1, 2hM]$.

Proof. By symmetry, it suffices to prove one of the two assertions; we choose the second. It suffices to show that, if $n \in B$ and $n < 2hM$, then $n+1 \in B$. In the window $W = [n+1-h, n+1+h]$ centered at $n+1$, all the points to the left of $n+1$ are also in the window $[n-h, n+h]$ centered at n and are distinct from $\mu(n)$. They therefore have F-values smaller than $F(\mu(n))$. Furthermore $\mu(n) \in W$. So the points of W to the left of $n+1$ cannot serve as $\mu(n+1)$. □

The argument in this proof extends easily to show that, when $n \in B$ and $n < 2hM$, then $\mu(n+1)$ is either $\mu(n)$ or $n+1+h$, whichever has the larger F-value.

The lemma shows that an interval $[1, 2hM]$ without local maxima can be split into two subintervals that are without local maxima in a stronger, one-sided sense. In A, every element n is prevented from being a local maximum by something to its left in its window (i.e., something in $[n-h, n-1]$, namely $\mu(n)$), while in B, everything is prevented from being a local maximum by something to its right (i.e., in $[n+1, n+h]$).

7.4.2. *Greedy Increasing Sequence.* We temporarily confine our attention to the subinterval B of $[1, 2hM]$ where (for our still fixed F) we have, for each n, a $\mu(n) \in [n+1, n+h]$ with a larger F-value than n has. (To avoid possible confusion, we note that $\mu(n)$ need not be in B; it could be larger than $2hM$.) Of course, what we do with B can also be done symmetrically with A.

Let (g_k) be the $\rightarrow$greedy increasing sequence in B for (the restriction to B of) the file F, as defined in Section 2.4. Since g_{k+1} is defined as the smallest $n > g_k$ with $F(n) > F(g_k)$ and since, by definition of B, $\mu(g_k)$ is such an n, we have

$$g_{k+1} \leq \mu(g_k) \leq g_k + h.$$

Thus, the greedy increasing sequence increases in steps of at most h and therefore has at least $\lceil |B|/h \rceil$ terms. (We can round $|B|/h$ up to an integer, rather than down, because g_0 is the first element of B and therefore g_k is no larger than the $kh+1^{\text{st}}$ element of B.) Notice that this fact makes $F \restriction B$ quite atypical. Indeed, as we saw in Section 2.4 the expectation of the length of the greedy sequence in an interval of size $|B|$ is approximately $\ln |B|$. So the greedy sequence for F is far longer than expected when $|B|$ is sufficiently large compared to h.

Recall from Proposition 12 that the elements g_k of the greedy sequence are exactly the $\rightarrow$maxima of F in B.

7.4.3. *Good Cuts.* By a *cut* in $[1, 2hM]$, we mean a partition of $[1, 2hM]$ into two subintervals A' and B', with A' lying to the left of B'. That is, $A' = [1, c]$ and $B' = [c + 1, 2hM]$, for some $c \in [0, 2hM]$; this allows the possibility that A' or B' could be empty. We think of this cut as being located between c and $c+1$, and so we say that it lies just to the right of c and just to the left of $c+1$. We also use terminology like "consecutive cuts" in the same sense.

A cut (A', B') will be called

- *right-good* if B' has at least one $\rightarrow$maximum in every h consecutive elements,
- *left-good* if A' has at least one $\leftarrow$maximum in every h consecutive elements, and
- *good* if it is both left- and right-good.

Our results above show that, when F has no local maximum in the interval $[1, 2hM]$, this interval admits at least one good cut, namely the partition into the specific pieces A and B defined above. It will be useful to know that, in fact, there are usually several good cuts

Lemma 59. *At least one of the following three statements is true.*

1. *The cut* $(\varnothing, [1, 2hM])$ *is good.*
2. *The cut* $([1, 2hM], \varnothing)$ *is good.*
3. *There are at least* $h + 1$ *consecutive cuts, all of which are good.*

Proof. The cut $(\varnothing, [1, 2hM])$ is vacuously left-good, so if it is right-good then we have the first alternative of the lemma. So we assume for the rest of the proof that $(\varnothing, [1, 2hM])$ is not right-good. Symmetrically, we assume that $([1, 2hM], \varnothing)$ is not left-good.

If a cut $([1, c], [c+1, 2hM]) \neq ([1, 2hM], \varnothing)$ is right-good, then so is the next cut to the right, $([1, c + 1], [c + 2, 2hM])$, and therefore, by induction, so are all cuts further right. The reason is that each $\rightarrow$maximum for $[c + 1, 2hM]$ except $c + 1$ is also a $\rightarrow$maximum for $[c + 2, 2hM]$. (Note that a $\rightarrow$maximum for $[c + 2, 2hM]$ need not be a $\rightarrow$maximum for $[c + 1, 2hM]$, because its F-value may be smaller than $F(c + 1)$.) Let p be the largest number in $[1, 2hM]$ such that the cut just to the left of p is not right-good; this exists because the cut just to the left of 1, i.e., the cut $(\varnothing, [1, 2hM])$, is not right-good. Since right-goodness is preserved when one moves a cut to the right, we see that the right-good cuts are exactly those that are to the right of p. Because our original cut (A, B) is good, we know that $p \in A$.

Similarly, let q be the smallest number in $[1, 2hM]$ such that the cut just to the right of q is not left-good, and observe that $q \in B$. In particular, $p < q$. Also note that the left-good cuts are exactly those that are to the left of q. Therefore, the good cuts are those that lie between p and q. To show that there are at least $h + 1$ of these, suppose not. That means $q \leq p + h$.

We assume, for the rest of the proof, that $F(p) < F(q)$. This entails no loss of generality, because the other case, $F(q) < F(p)$, can be treated symmetrically.

Consider the $\rightarrow$greedy increasing sequence in the interval $[p, 2hM]$. It begins with $g_0 = p$, and its next term g_1 is the first $n \in [p + 1, 2hM]$ such that $F(p) < F(n)$. Now q is such an n, so we have $g_1 \leq q \leq p + h$. That is, the difference between the first two elements of this greedy sequence is at most h.

All $\rightarrow$maxima for $[p + 1, 2hM]$ that are to the right of g_1 are also $\rightarrow$maxima for $[p, 2hM]$. Indeed, the only way a $\rightarrow$maximum n for $[p+1, 2hM]$ could fail to be a $\rightarrow$maximum for $[p, 2hM]$ is to have $F(n) < F(p)$. But if $n > g_1$, then its being a $\rightarrow$maximum for $[p + 1, 2hM]$, which contains g_1, implies that $F(n) > F(g_1) > F(p)$. So n cannot fail to be a $\rightarrow$maximum for $[p, 2hM]$. Now the difference between any two consecutive $\rightarrow$maxima for $[p + 1, 2hM]$ is at most h because the cut just to the left of $p + 1$ is right-good (by definition of p). So the difference between consecutive $\rightarrow$maxima for $[p, 2hM]$ to the right of g_1 is at most h also. This fact, together with the result of the preceding paragraph, shows that the cut just to the left of p is right-good, contrary to the definition of p. This contradiction (together with the analogous contradiction, to the definition of q, when $F(p) > F(q)$) completes the proof of the lemma. □

Corollary 60. *There is a good cut* (A', B') *such that the cardinalities* $|A'|$ *and* $|B'|$ *are divisible by* h.

Proof. The conclusion of the corollary is obvious if either of the first two alternatives in the lemma holds. Under the third alternative, we have $h+1$ consecutive good cuts (we actually need only h), so one of them must be at a distance from the left end that is divisible by h. That is,

some good cut (A', B') has $|A'|$ divisible by h. $|B'|$ has the same divisibility property because $2hM$ does. □

We summarize the preceding work as follows.

Proposition 61. *For any $F \in PFE^{\mathbb{Z}}$ (without ties) that has no local maximum in $[1, 2hM]$, there exists a c, with $0 \le c \le 2M$, such that the cut $([1, ch], [ch+1, 2hM])$ is good.*

7.4.4. *Probabilities of →Maxima.* We now un-fix F; that is, F will now be a random element of $\mathrm{PFE}^{\mathbb{Z}}$. Our ultimate goal is to estimate, from above, $\mathbf{Prob}(F$ has no local maximum in $[1, 2hM])$. In view of the results obtained above, we begin by estimating, for a fixed c, the probability that the cut $([1, ch], [ch+1, 2hM])$ is right-good, i.e., that every h consecutive elements of $[ch+1, 2hM]$ include at least one →maximum. There will be a similar estimate for the probability that $([1, ch], [ch+1, 2hM])$ is left-good, and afterward we shall combine these estimates and sum over all c to estimate the probability that there is no local maximum in $[1, 2hM]$.

Recall from Propositions 14 and 17 that, as n ranges over the interval $J = [ch+1, 2hM]$, the events "n is a →maximum" are probabilistically independent and their probabilities are given by $1/(n - ch)$. We can now easily compute the probability that a given subinterval of length h in $[ch+1, 2hM]$, say $[ch+a+1, ch+a+h]$, contains a →maximum of J. Indeed, the probability that this interval contains no →maximum for J is given by a telescoping product:

$$\prod_{i=1}^{h}\left(1 - \frac{1}{a+i}\right) = \prod_{i=1}^{h}\frac{a+i-1}{a+i} = \frac{a}{a+h}.$$

So the complementary probability, that there is at least one →maximum of J in $[ch+a+1, ch+a+h]$, is $h/(h+a)$.

Break the interval $[ch+1, 2hM]$ into $2M - c$ subintervals, which we call *blocks*, of length h. The event that the cut $([1, ch], [ch+1, 2hM])$ is right-good is included in the event that each of these blocks contains a →maximum of J, and the probability of the latter event can be computed by combining the computation in the preceding paragraph with the independence result in Lemma 17. Numbering the blocks from 1 to $2M - c$, we can apply the computation from the preceding paragraph, with $a = (j-1)h$, to see that the jth block contains a →maximum of J with probability $h/(h + (j-1)h) = 1/j$. Thus, by independence, the probability that every block contains a →maximum of J is

$$\prod_{j=1}^{2M-c}\frac{1}{j} = \frac{1}{(2M-c)!}.$$

This probability therefore provides an upper bound for the probability that the cut $([1, ch], [ch+1, 2hM])$ is right-good.

Similarly, the probability that the cut $([1, ch], [ch+1, 2hM])$ is left-good is bounded above by $1/c!$. Furthermore, all the events "n is a →maximum of $[ch+1, 2hM]$ are independent of the events "m is a ←maximum of $[1, ch]$". This is because the former events refer only to the relative ordering of values of F at places $> ch$ while the latter refer only to the relative ordering values of F at places $\le ch$. Therefore, the probability that $([1, ch], [ch+1, 2hM])$ is good is bounded above by

$$\frac{1}{(2M-c)!}\cdot\frac{1}{c!} = \binom{2M}{c}\frac{1}{(2M)!}.$$

The event that $[1, 2hM]$ contains no local maximum is included in the union of the events that $([1, ch], [ch+1, 2hM])$ is good, where c ranges from 0 to $2M$. Thus, the probability of no local maximum in $[1, 2hM]$ is bounded above by

$$\sum_{c=0}^{2M}\binom{2M}{c}\frac{1}{(2M)!} = \frac{2^{2M}}{(2M)!}.$$

7.4.5. *Comparison With Other Methods.* We compare the probability of unpleasantly long chunks under the point filter, LBFS, interval filter, and local maximum chunking methods. Recall from the preceding calculations that, when the parameters of all three methods are adjusted to produce the same average chunk length $2h+1$, the probabilities of finding no cutpoints in the interval $[1, 2hM]$ (approximately M average chunks) are

$$
\begin{aligned}
&> e^{-M} && \text{for the pure point filter method}\\
&> e^{-(k+1)M/(1-k/2h)} \approx e^{-(k+1)M} && \text{for the LBFS method}\\
&> Ae^{-BM} && \text{for the interval filter method}\\
&< 2^{2M}/(2M)! && \text{for the local maximum method,}
\end{aligned}
$$

where in the case of LBFS $k = h/c$ is the expected number of candidates in a segment of length equal to the horizon, and where in the case of the interval filter method $1 < B < e$.

Notice that the pure point filter, LBFS, and interval filter methods give probabilities that decrease "only" exponentially as $M \to \infty$, while the local maximum method gives a probability that decreases more rapidly. This can be seen by comparing the logarithms of the probabilities, using Stirling's approximation for the factorial. The four logarithms are $-M$, $-(k+1)M$, $-BM + \ln A$, and asymptotically $-2M(\ln M - 1)$, respectively. The last is, for large enough M, much smaller (i.e., more negative) than the others because of the $\ln M$ factor.

Unfortunately, this comparison of behaviors as $M \to \infty$ can be misleading. The reason is that our computation for the local maximum method assumed that there are no ties, i.e., that no file entry is repeated in the interval $[1-h, 2hM+h]$. That assumption is reasonable as long as $(2hM)^2 \ll |\text{PFE}|$, but not as $M \to \infty$; in fact the assumption is obviously false once M is large enough.

So a true comparison should use relatively small values of M. These are also the values of M that are relevant for practical purposes. We would like to have small probabilities for cut-less intervals of length, say, 5 average chunk lengths. The corresponding probabilities for very large M will be too small to worry about.

It turns out that, once $M \geq 4$, our upper bound $2^{2M}/(2M)!$ for the local maximum method is smaller than the lower bound e^{-M} for the pure point filter method and also the lower bound $e^{-5M/4}$ for the LBFS method when $k = 1/4$. If we increase M to 7, the local maximum method gives a probability smaller by almost a factor 1000 than the LBFS method with $k = 1/4$.

Remark 62. Our upper bound for the probability of no local maximum in $[1, 2hM]$ is rather rough; we sacrificed a good deal of information by using only the fact that each of the $2M - c$ blocks contains a $\rightarrow$maximum, when in fact every interval of length h in J must contain a $\rightarrow$maximum. The reason for this sacrifice is to obtain independence and thereby facilitate the computation. Because the blocks are pairwise disjoint, the events that they contain $\rightarrow$maxima of J are independent. If we used all subintervals of J of length h, rather than only the blocks, we would lose the disjointness and thus the independence. If the computation could be completed despite this loss, it would surely yield a tighter upper bound than the one we obtained.

8. Computing Local Maxima

In this section, we discuss ways of finding the local maxima in a (finite) file. For the other chunking methods that we have discussed — point filter, LBFS, and interval filter — it is clear that the cutpoints of a file F can be found in a single pass through the file, performing some elementary test (divisibility by a given c or membership in U) on each file entry, and counting (to see whether we are at a blocked location for LBFS or whether we have matched an interval filter). The most obvious algorithm for determining local maxima, namely to compare each file entry with each of the $2h$ others in its window, is far less efficient, as it requires $2h$ operations per file position. Fortunately, there are better algorithms. We shall describe two of them. One is rather straightforward and needs just two comparisons per file entry. That is, the total number of comparisons needed is no more than twice the length of the file. The second algorithm is more

sophisticated and needs, on average, only

$$1 + \frac{\ln h}{h} + O\left(\frac{1}{h}\right)$$

comparisons per file entry. Since h is large, this amounts to barely more than 1 comparison per file entry, so the local maximum chunking method does not require significantly more work than others to find the cutpoints.

Remark 63. For operation on a modern CPU, the important metric is the number of branch mispredictions encountered during chunk computation. Modern CPUs use advanced branch prediction hardware to opportunistically continue computation assuming a branch predicate evaluates to the same value as it did when it was previously encountered. This pays off as long as branch predicates are biased to evaluate to the same value, but is of no help if branch predicates evaluate to different truth values in random alternation. Counting the number of comparisons per file entry provides an upper bound on branch mispredictions, and is therefore a good abstract measure for the running time of the chunking methods.

We begin by describing the more straightforward of the two algorithms. The idea is to read the file F, from left to right, producing a list of the local maximum positions, and keeping track of the additional information that is relevant to the computation of later local maxima. More precisely, the algorithm will keep track of pairs $(i, F(i))$ that might affect future decisions about what is or is not a local maximum. Those pairs are of two sorts.

First, there are the pairs $(i, F(i))$ that might turn out to be local maxima, but are not yet known to be local maxima. This means that $F(i)$ is larger than the h immediately preceding values of F as well as the subsequent values that have been read so far, but the number of these subsequent values is $< h$. So i looks as though it could be a cutpoint but some values in its window remain to be read, so it may yet turn out not to be a cutpoint. Notice that pairs of this sort must always have i within the last h positions that have been read; once we read farther than that, we will know whether i is a cutpoint, so it will either be put on the output list of local maxima or dropped from consideration.

Second, there are the pairs $(i, F(i))$ that might prevent some position that we haven't yet visited from being a cutpoint. That is, we might in the future read $F(j)$ at position j and decide that j cannot be a cutpoint because $F(i) \geq F(j)$. In principle, any of the pairs $(i, F(i))$ among the last h that were read could play this role, but many of them can safely be ignored. Specifically, suppose that we have read a larger value at a later position, say $F(i') \geq F(i)$ with $i' > i$. Then any future j that is prevented from being a cutpoint by $(i, F(i))$ is also prevented by $(i', F(i'))$. Indeed, we have $F(i') \geq F(i) \geq F(j)$ and, if i is in the window of j, then so is i' because $i < i' < j$.

This means that the only i's for which we have to remember $(i, F(i))$ because it might prevent a future j from being a cutpoint are those i, within the last h positions read, for which $F(i)$ is larger than all later F-values already read. Proposition 12, applied in the right-to-left direction, tells us that these values of i constitute the $\leftarrow$greedy sequence in the interval of the last h positions read.

Notice that, if there is an $(i, F(i))$ of the first sort, a candidate for being a cutpoint, then it is also of the second sort, since it is within the last h positions read and its F-value is larger than the later ones. Summarizing, we see that our algorithm should maintain the following information as it reads through the file F:

- $(i, F(i))$ for i in the $\leftarrow$greedy subsequence of the interval of the last h positions read, and
- one additional bit, telling whether the leftmost position in the greedy sequence (the one with the largest F-value) is a candidate for being a cutpoint, i.e., whether its F-value exceeds the h immediately preceding F-values.

Thus, our algorithm acts as follows while reading the file from left to right. It maintains a list Λ of pairs and a bit γ, and it (gradually) outputs a list of cutpoints. At position i, the algorithm performs the following steps, in the given order.

(1) Read $F(i)$.

(2) Go through Λ, in order, deleting any pairs $(j, F(j))$ with $F(j) \leq F(i)$. Stop when and if a pair is not deleted.
(3) If $(i-h, F(i-h))$ is in Λ, add it to the output list of cutpoints and set $\gamma := 0$.
(4) Add $(i, F(i))$ to the beginning of Λ and, if Λ has no other elements, set $\gamma := 1$.
(5) Delete $(i-h, F(i-h))$ from Λ (if it's present).

When the algorithm has finished processing position i in this manner, Λ contains $(j, F(j))$ for j in the $\leftarrow$greedy sequence for $[i-h+1, i]$, γ is 1 if and only if the last (leftmost) element of Λ is still a candidate to be a cutpoint, and the output produced so far consists of all the cutpoints at positions $\leq i-h$. Note that the algorithm maintains the property that the pairs $(j, F(j))$ in Λ always occur in order of decreasing j and increasing $F(j)$. This is why the sentence beginning "Stop" in instruction (2) is justified; the elements of Λ that are not inspected have $F(j)$ at least as large as the non-deleted one that triggered the stop, and so they should also not be deleted.

To estimate the number of comparisons performed (while executing instruction (2)) during a run of this algorithm, we associate to each comparison a position in the file as follows. If the comparison of the newly read $F(i)$ with an earlier $F(j)$ results in the deletion of $(j, F(j))$ from Λ (because $F(j) \leq F(i)$), then associate position j to this comparison; we refer to this as "association with deletion". Otherwise, i.e., if $F(j) > F(i)$, then associate position i to the comparison; we refer to this as "association without deletion".

Since any $(j, F(j))$ enters Λ just once, it is deleted at most once, and so j has at most one comparison associated to it with deletion. Furthermore, because of the "stop" part of instruction (2), each i has at most one comparison associated to it without deletion. So altogether, each position has at most two comparisons associated to it. Thus, the total number of comparisons performed by the algorithm is at most twice the length of the file.

Remark 64. We can be more precise about the number of comparisons. The only way a position i can avoid having a comparison associated to it with deletion is to have $F(i) > F(j)$ for all $j > i$ in its window. Call such a point a right semi-maximum. The only way i can avoid having a comparison associated to it without deletion is to have $F(i) \geq F(j)$ for all $j < i$ in its window. Call such a point a left semi-maximum. (Note the asymmetry: right semi-maxima satisfy a strict inequality and left semi-maxima only a non-strict one. Of course, this doesn't matter if PFE is big enough and we ignore ties.) Thus, the number of comparisons performed by this algorithm is twice the length of the file, minus the sum of the number of right and left semi-maxima. Since a local maximum is both a right and a left semi-maximum, it follows that the number of comparisons is at most twice the number of positions that are not local maxima.

We now turn to a more sophisticated algorithm, which, compared to the preceding one, cuts the number of comparisons almost in half, on average.

The algorithm splits the file into blocks of length $h+1$, and it processes the blocks in order, from the leftmost to the rightmost. For brevity, we ignore the trivialities arising if the length of the file isn't exactly divisible by $h+1$.

For a position i to be a local maximum, it is necessary (but not sufficient) that $F(i) > F(j)$ for all j in the block containing i, because all such j are in the window $[i-h, i+h]$ centered at i. We call i a *candidate* if it fulfills this necessary condition. For a candidate to be a local maximum it must, in addition, have an F-value greater than that of any position within h in the immediately previous and immediately following blocks. If the algorithm finds that a candidate fails to satisfy this additional requirement, we shall say that it *kills* the candidate; we use the phrase *live candidate* to mean a candidate that has not (yet) been killed.

When processing a block B, the algorithm will produce the following information:

(1) If, when it starts processing B, there is a live candidate in the immediately previous block, it will decide whether that candidate is a local maximum.
(2) It will produce the $\leftarrow$greedy sequence of B, in decreasing order of positions (and thus increasing order of F-values).
(3) It will decide whether the last term in the $\leftarrow$greedy sequence is a candidate.
(4) If the last term is a candidate, it will decide whether it is to be killed because of a larger or equal F-value in the previous block (and within the candidate's window).

Recall that, by Proposition 12 (applied with right and left reversed), the $\leftarrow$greedy sequence in item (2) consists of those positions in B where the F-value is greater than all later F-values in B. It can thus be found by reading the F-values in the block B, from right to left, keeping track of the highest value seen so far, and adding to the $\leftarrow$greedy sequence any position where the newly read value exceeds the largest previously seen value. Notice that the algorithm processes the blocks in left-to-right order but processes the positions within any block in right-to-left order.

In connection with item (3), notice that the last element of the $\leftarrow$greedy sequence will always have an F-value $\geq$ all other F-values for the block B. But for a candidate, we need strict inequality here. So item (3) amounts to checking for ties for the highest F-value in the block.

Most of the algorithm's work goes into item (2), so we begin our description there. Notice that, if we were just producing the $\leftarrow$greedy sequence (and not doing anything about items (1), (3), and (4)), this would involve h comparisons. Each position in the block, except the rightmost, must have its F-value compared, at the time the algorithm reads it, with the F-value of the currently last element of the sequence under construction. So this task requires approximately one (exactly $\frac{h}{h+1}$) comparison per position in the file.

Item (3) can be handled simultaneously with the construction of the $\leftarrow$greedy sequence. Whenever a position is put into the sequence, call it a candidate (tentatively). If another position is put in later (because it has a larger F-value) kill the old candidate while making the newly added position a candidate. Also, if another position in the block is found to have the same F-value as the current candidate, then kill the candidate (even though it remains in the $\leftarrow$greedy sequence and the later position with the same F-value is not added to the sequence). Thus, item (3) requires no additional comparisons.

Let us refer to the process just described, running through B in reverse order to handle items (2) and (3), the *ordinary* run through B. If, when we start processing block B, there is no live candidate in the immediately previous block, then the ordinary run through B handles item (1) vacuously and we need only consider item (4), which we shall do later. But if there is a live candidate in the preceding block, then the ordinary run must be modified in order to handle item (1), and we now describe this modification.

Suppose, therefore, that position m is a live candidate in the block just before B. Being a candidate, it has a larger F-value than all other elements of its block. Furthermore, being live, it has a larger F-value than all positions to its left in its window, i.e., in $[m-h, m-1]$, because otherwise it would have been killed during the processing of its own block — see item (4). So it will be a local maximum unless there is a larger or equal F-value at a position that is in B and $\leq m+h$. Our task is to detect such an F-value, if there is one, and then kill m. And we must do this without excessive comparisons of F-values.

Begin processing B by the ordinary run until you reach position $m+h$. (The point is that, until this moment, you're working with positions outside the window of m and thus irrelevant to item (1).) When you reach $m+h$, there is a branching according to whether the current last position in the $\leftarrow$greedy sequence, say g, has $F(g) \geq F(m)$ or $F(g) < F(m)$.

Suppose first that $F(g) \geq F(m)$. (To avoid confusion, notice that this case hypothesis doesn't kill m because g is beyond the right end of the window of m.) In this case, continue going leftward through the block B, but, instead of comparing F-values with $F(g)$ (as the ordinary run would), compare them with $F(m)$. As long as they are $< F(m)$, they can be ignored as they don't kill m and they don't go into the $\leftarrow$greedy sequence (because $F(g) \geq F(m)$). If you find an F-value equal to $F(m)$, then that kills m, but the position still doesn't go into the $\leftarrow$greedy sequence; once m is killed, resume the ordinary run from the next position on the left. Finally, it you find an F-value strictly greater than $F(m)$, then kill m, and resume the ordinary run from the current position. Notice that, in this last situation, one position has its F-value compared with both $F(m)$ and $F(g)$. So the total number of comparisons will not be h as computed above for the ordinary run but $h+1$; that's still an average of only one comparison per position in B.

Now consider the other case, where $F(g) < F(m)$. In this case, continue with the ordinary run but, whenever a position g' is added to the $\leftarrow$greedy list, check whether $F(g') \geq F(m)$. If so, then kill m and continue with the ordinary run and no further comparisons with $F(m)$. If, on the other hand, $F(g') < F(m)$, then m remains live and the next addition to the $\leftarrow$greedy sequence will also need a comparison with $F(m)$. Notice that, if m should be killed, then this

procedure will kill it. Specifically, if m should be killed, then there is an x in its window and in B with $F(x) \geq F(m)$, because, as noted above, any other reason for killing m would have done so while the previous block was processed. The rightmost such x will be added to the $\leftarrow$greedy sequence, will therefore have $F(x)$ compared to $F(m)$ by our algorithm, and will thus kill m.

In the case just considered, it is possible for many positions to have their F-values compared with the F-values at both m and the current last position in the $\leftarrow$greedy list. Indeed, this happens whenever a position $\leq m+h$ is added to the $\leftarrow$greedy list as long as m remains live. A priori, the number of such occurrences is bounded by the length of the $\leftarrow$greedy list, which is, as we saw in Section 2.4, on average approximately $\ln h$. In fact, we shall get a much better bound, on average, later, but first we finish the description of the algorithm by showing how to handle item (4).

Item (4) is handled by a separate process after the completion of the run through B (either the ordinary run or the modification to handle item (1)). Suppose, therefore that this run has been completed and that it resulted in a candidate, namely the last term g of the $\leftarrow$greedy sequence, and that this candidate hasn't been killed yet (i.e., no point to its left in B had the same F-value). For item (4), we must check whether some point x in the window of g and in the previous block had a larger F-value, $F(x) > F(g)$. The key observation is that, if this happens, then the rightmost such x is in the $\leftarrow$greedy sequence of the previous block. Indeed, for all $y > x$ in that block, we have $F(x) > F(g) \geq F(y)$. So to look for such an x, it suffices to look through the terms of the previously computed $\leftarrow$greedy sequence of the preceding block. Since the length of the $\leftarrow$greedy sequence in any block is, on average, only approximately $\ln h$, we can handle item (4) with only $\ln h$ additional comparisons, i.e., only $\frac{\ln h}{h}$ comparisons per position, on average.

This completes the description of the algorithm and most, but not all, of the estimation of how many comparisons of F-values are needed. On average, we have, during the processing of a block, at most

- h comparisons to handle items (2) and (3),
- $\ln h$ comparisons to handle item (1) in the case where $F(g) < F(m)$ and $F(g') < F(m)$ for many subsequent elements g' of the $\leftarrow$greedy sequence, and
- $\ln h$ comparisons to handle item (4).

We now show that the $\ln h$ associated to item (1) can be reduced greatly, namely to a constant. Thus, the final estimate for the average number of comparisons will be $h + \ln h + O(1)$ per block or $1 + \frac{\ln h}{h} + O(\frac{1}{h})$ per position. Notice that, even without this improvement, we already know that the average number of comparisons per position is at most $1 + \frac{2\ln h}{h}$.

Remark 65. Before proceeding with the proof, we give a rough, intuitive argument for why we might expect the number of comparisons needed for (1) to be less than the estimate $\ln h$. If there is a live candidate in the preceding block, its expected position is at the middle of that block, and so our algorithm begins paying attention to item (1) around the middle of the current block. Once it begins paying attention, it performs an extra comparison when adding an element to the $\leftarrow$greedy sequence. But, if attention begins at the middle of the block, then there is a 50% chance that nothing will be added to the $\leftarrow$greedy sequence from that point on, because there is a 50% chance that the largest F-value in the block is in the right half of the block, in which case the $\leftarrow$greedy sequence is already complete when the algorithm reaches the middle of the block. Of course, a real proof must take into account that the live candidate in the preceding block isn't known to be at the midpoint, so its variability must be taken into account. That is what the following argument does.

We wish to bound, from above, the expectation of the number X of additional comparisons introduced by the part of the algorithm that handles item (1). To begin, we imagine some changes in the algorithm, which make X worse, i.e., bigger, but which simplify the computation of its expectation. Of course by bounding the expectation of the imagined, larger X, we obtain a fortiori the same bound for the expectation of the actual X.

The first imagined change is that we pretend that the position m of a maximum of F in the preceding block is a live candidate, whether or not it really is one. This clearly increases the number X of extra comparisons, because we will be doing comparisons for the sake of item (1) in

some cases where the actual algorithm can ignore item (1) because there is no live candidate. In the case where the maximum is attained at several positions, we choose one of them at random to serve as m. (This is one of the cases where there is really no live candidate.) Observe for future reference that m is equally likely to be any of the h positions in the preceding block, and therefore the place where our algorithm will begin doing extra work is equally likely to be immediately to the left of any of the $h+1$ positions in the current block B.

The second imagined change is that, once we get into the range where the extra work is done, every element added to the $\leftarrow$greedy sequence contributes an extra comparison. This makes X worse because in the actual algorithm the extra comparisons would end when and if the $\leftarrow$greedy sequence acquires a member g with $F(g) \geq F(m)$.

With both imagined changes, X is simply the number of positions that are put into the $\leftarrow$greedy sequence and are $\leq m+h$. As noted above, m is a random variable uniformly distributed in the preceding block, and so $m+h$ is uniformly distributed in the range from the rightmost element of the preceding block to the next-to-rightmost element of the current block B. (In other words, the range of values of $m+h$ is B shifted one step to the left.) Let us write r for the location of $m+h$ in this range, but counted from right to left (the direction that the algorithm goes while processing B). Thus, $r=1$ means the next-to-rightmost element of B; $r=h+1$ means the rightmost element of the preceding block. All $h+1$ values of r are equally likely. For any particular r, the elements of the $\leftarrow$greedy sequence that contribute to X are those whose distance from the right end of B is $> r$. (For example, if $r=1$ then all but the rightmost element of B can contribute to X, while if $r=h+1$ then nothing contributes to X.) The position at distance j from the right end of B has probability $1/j$ of being in the $\leftarrow$greedy sequence; see Section 2.4. Therefore, the expectation of X (as increased by our imagined changes) is

$$\frac{1}{h+1}\sum_{r=1}^{h+1}\sum_{j=r+1}^{h}\frac{1}{j}.$$

This double sum is easy to estimate by interchanging the order of summation. Specifically, for any fixed j in the relevant range $2 \leq j \leq h$, the fraction $\frac{1}{j}$ occurs $j-1$ times, namely once for each r in the range $1 \leq r \leq j-1$. So these terms contribute $\frac{j-1}{j} < 1$. This happens for each of the $h-1$ values of j, so the double sum is $< h-1 < h+1$, and the expectation of X is therefore < 1.

9. Evaluation and Experiments

The chunking methods were at first developed and tested without a theoretical analysis. As part of the initial development of RDC we conducted several measurements on different chunking methods. Many of the experiments are reported in [28]. One immediately recognized advantage with the local-maximum chunking method was that it only required to be configured with a minimum chunk length, while the point-filter method required both a minimum length and a cut-point probability (in the form of a bit-mask). However, the best configuration could potentially be identified once and for all, making this advantage irrelevant. A more important advantage was soon recognized experimentally: the local-maximum chunking method produced higher-quality compression. Eventually we set out to develop a theoretical analysis of chunking methods whose results are reported above. In this section, we report some experimental results.

In the experiments we measured the effect of chunking methods as well as other aspects of RDC on a large corpus of internal Microsoft design documents as well as hard-disk file images (known as virtual hard-disks). The design documents were mostly in the Microsoft Office Word, Power-Point and Excel formats. This corpus of around 10,000 Microsoft Office files was chosen to represent user scenarios for the RDC protocol. We used a smaller set of around 20 virtual hard-disks for similar experiments. Each virtual hard-disk contains from a few hundred MB up to 10 GB of data. The experiments exercised RDC when the receiver could use chunks from several other local files in addition to the file being transmitted; the use of multiple local files allows RDC to reduce bandwidth usage. The initial evaluation therefore covered much more than just the isolated effect of chunking methods. Nevertheless, some of the experiments have specifically covered the efficiency of the chunking methods, and we recall and present some of the results

in the next subsection. We then describe a few selected measurements that are targeted more narrowly on evaluating the chunking methods.

9.1. **Computation overhead.** While developing the RDC protocol we were mainly interested in measuring the combined overhead of chunking and other factors dominating file transfer. And we measured the combined client and server overhead for file transmission. We studied in particular the following two extreme scenarios. In one scenario, the client happens to have the exact file being transmitted, and in the other scenario the client has none of the chunks being transmitted. We compared RDC against RSYNC and two other widely available utilities for differential compression, namely xdelta [19], and BSDiff [21]. (The LBFS system is less appropriate for such an experiment because normally it relies on a database that contains the chunks of multiple files.) On a Pentium 3 machine we measured an average number of cycles per byte of the transmitted file. In the first scenario we had 31 cycles for RDC, 45 for RSYNC, 39 for xdelta, and 2580 for BSDiff. In the second scenario, we had 36 cycles per byte for RDC, 32 for RSYNC, 410 for xdelta and 2780 cycles for BSDiff. Thus, RDC and RSYNC appear comparable as far as the CPU overhead is concerned, while the two other utilities require much more CPU. (In fact they require more memory as well.)

Early performance testing of the RDC protocol indicated that calculating chunk boundaries and signatures (that is the hash values of the chunks) is a significant CPU bottleneck. Low-level machine-architecture specific optimizations were used, with significant benefit, to boost performance of the inner loops in the chunking routines.

We also used the optimization described below in Section 10.2. That optimization results in a chunking algorithm that is hash-less in the sense that it bypasses the rolling hash stage for the local-maximum chunking method. To compare the hash-less algorithm with the one based on a rolling hash we measured the average number of cycles required to compute the chunks. For the expected chunk length of 256 bytes, a 64 bit Pentium 4 machine requires 8.4 cycles per byte for the hash-less algorithm and 18.7 cycles per byte for the original local-maximum algorithm based on a fast (and low quality) rolling hash. Our fast rolling hash uses just two bitwise xor operations and one bitwise rotation per processed byte. The point-filter method cannot avoid the rolling hash stage but it takes advantage of branch predicting hardware while determining cut-points. As a result, it requires only 7.6 cycles per byte using the low quality hash. By definition low quality hashing has too many collisions. From the point of view of collisions, the hash-less method works as a perfect hash: there are no collisions. The point-filter method using the higher quality Karp-Rabin hash took 15.8 cycles per byte.

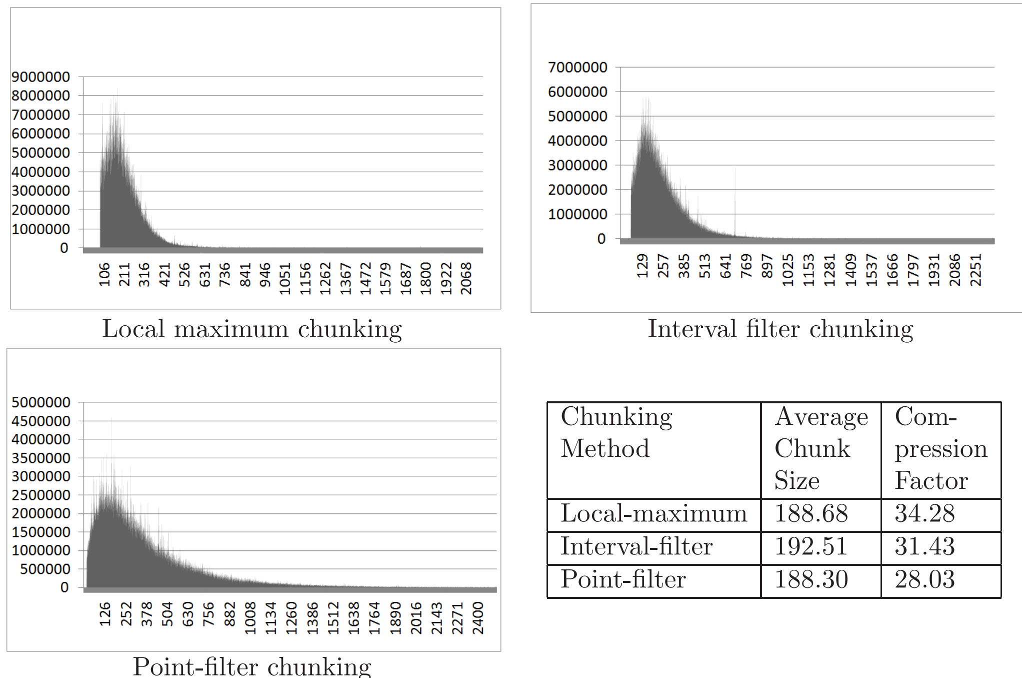
Overall, the overhead of computing chunk boundaries based on the local-maximum method seems reasonable for chunks of size 256 bytes and up. However we later noticed that the point filter method was preferable in scenarios (different from RDC scenarios) where chunks of size 30-40 bytes should be generated with as little CPU overhead as possible. In the context of such small chunks branch-predicting hardware is critical.

9.2. **Intra-file Compression.** In the following experiments we measure how the local maximum, the interval filter and the non-pure point-filter chunking methods (used in LBFS) compare. We observe the distribution of chunk sizes on real-world (non-random) files, and we measure the compressibility of long chunks as well as the effectiveness of the chunking methods for identifying repeated chunks in the same files.

Our experiments use three files from the RDC target domain. The first is the Outlook folder file (`.ost` file) of the first author. It contains 2 years of email messages and its size is 1.17 GB. The second file is a virtual hard disk (`.vhd` file) for Windows Server 2008, Enterprise edition. The file contains a full operating system installation and its size is 1.71 GB. Our third file is an 11 MB power-point presentation (`.ppt` file).

For the local-maximum chunking method we set the horizon size to 90 so that the expected chunk size is 181. For the interval-filter method we use a bit-mask with 6 bits so that the expected chunk size is $e \cdot 2^6 = 173$. And for the point-filter method we use a bit-mask with 7 bits and a minimal chunk length of 20 so that the expected chunk length is 148 bytes. However, as the experimental data show, the average chunk sizes vary depending on the files and tend to be slightly higher than the expected chunk sizes based on random files.

The effectiveness of finding repeated chunks in the same file is reflected in the overall compression ratio obtained by the methods. We observe that the local-maximum method finds up to 10% more repeated bytes than the other methods. Our graphs in Figures 3, 4 and 5 also give an indication of the distribution of chunk sizes. The horizontal line indicates chunk sizes. The vertical scale shows the total number of bytes used by chunks of the size indicated by the horizontal line.



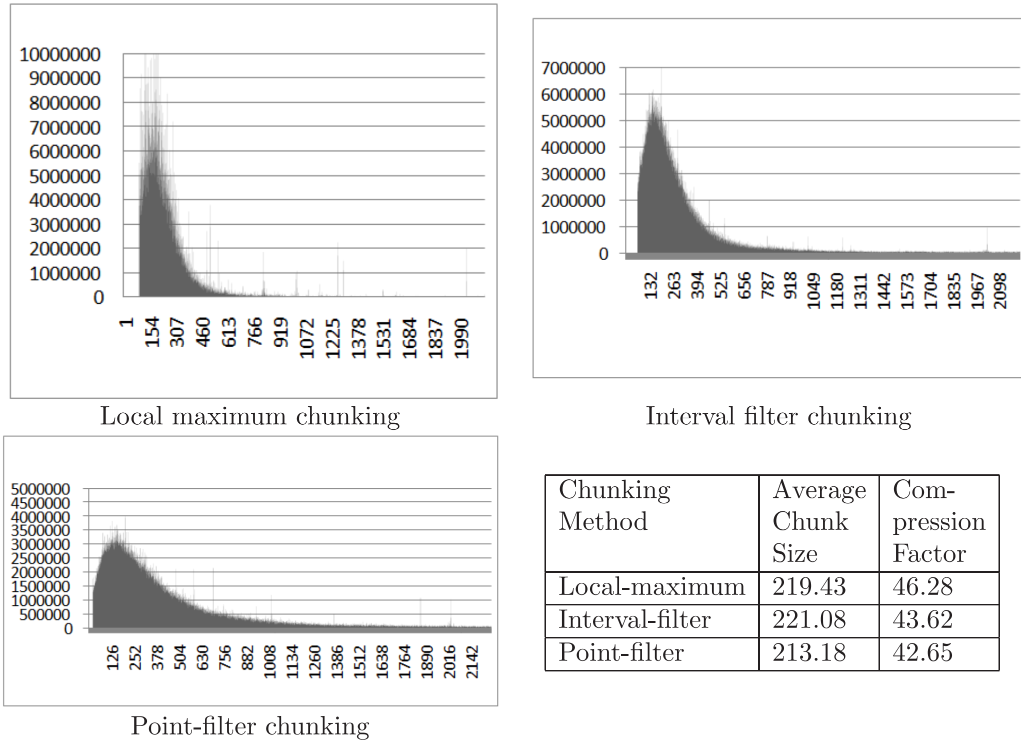
Chunking Method	Average Chunk Size	Compression Factor
Local-maximum	188.68	34.28
Interval-filter	192.51	31.43
Point-filter	188.30	28.03

FIGURE 3. Chunking-based compression of the Outlook folder file

9.3. **Distribution of Long Chunks.** In Section 5.4 we analyzed the probability of long chunks for the three chunking methods. As an experimental counter-part, Figure 6 summarizes how much of each file resides in chunks that are of size at most L, $2L$, $3L$, and up to $21L$ where L is the expected chunk length. In all three figures, the local-maximum method contains more chunk data close to the average chunk length; it is represented by the left-most and top-most line. For example, for the Outlook folder file, 43% of the chunks are of length at most L, and 90% of the chunks are of length at most $2L$. The second best lines in all three figures represent the interval-filter method. The lowest and right-most lines represent the point-filter method which has more file data in longer chunks. For example, for Outlook, to cover 90% of the file data requires including chunks of size up to $5L$ (as opposed to $2L$ in the local-maximum method).

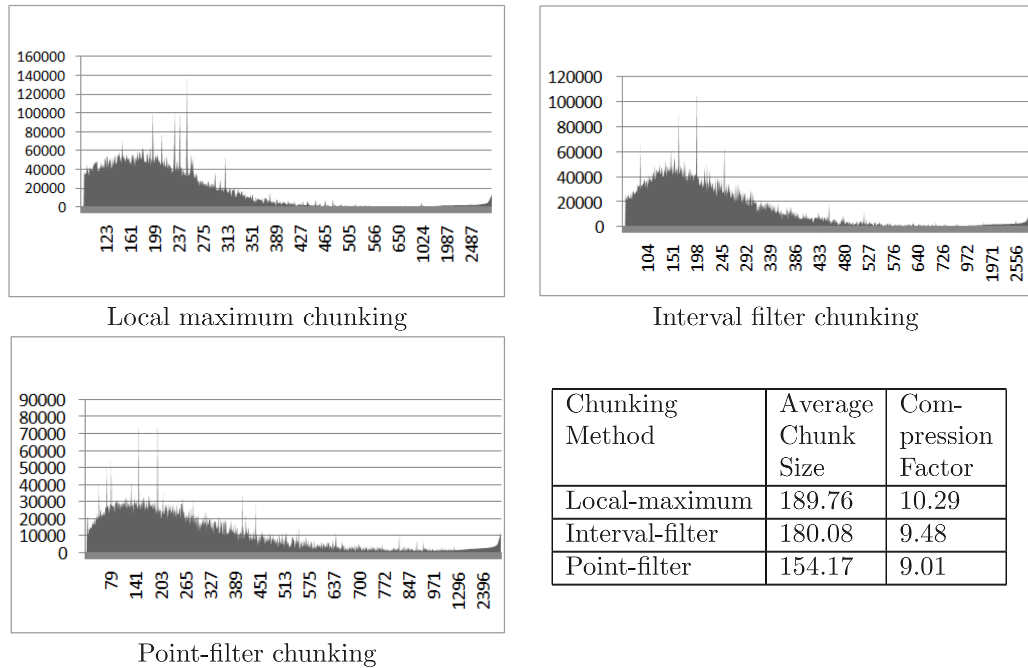
9.4. **Run-length compression.** For the local-maximum method and the Outlook file, we furthermore measured the compressibility of the chunks as a function of their size. We performed two experiments. The first was inspired by Remark 8. We used a simple run-length encoder that replaced repeated bytes and pairs of repeated bytes by a code. For example, it would replace the string $abababcccccdfg$ by $x3aby4cz3dfg$, where x is used as a control character for double-byte repetition, y for single byte repetition, and z is used otherwise. In the second experiment, we applied the standard Lempel-Ziv compression algorithm on the chunks to measure the compressibility more accurately.

Figure 7 summarizes the results. The x-axes of both histograms show the chunk lengths and the y-axis show the compression ratio. A ratio of 1 indicates the perfect compression. Note that practically all long chunks are highly compressed and that the run-length encoder is often sufficient for the long chunks.



Chunking Method	Average Chunk Size	Com-pression Factor
Local-maximum	219.43	46.28
Interval-filter	221.08	43.62
Point-filter	213.18	42.65

FIGURE 4. Chunking-based compression of the Windows Server virtual hard disk

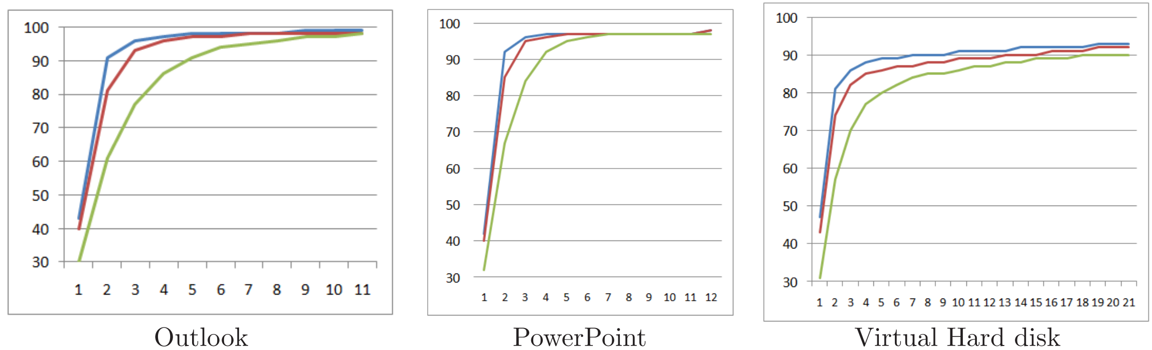


Chunking Method	Average Chunk Size	Com-pression Factor
Local-maximum	189.76	10.29
Interval-filter	180.08	9.48
Point-filter	154.17	9.01

FIGURE 5. Chunking-based compression of the Power-point file

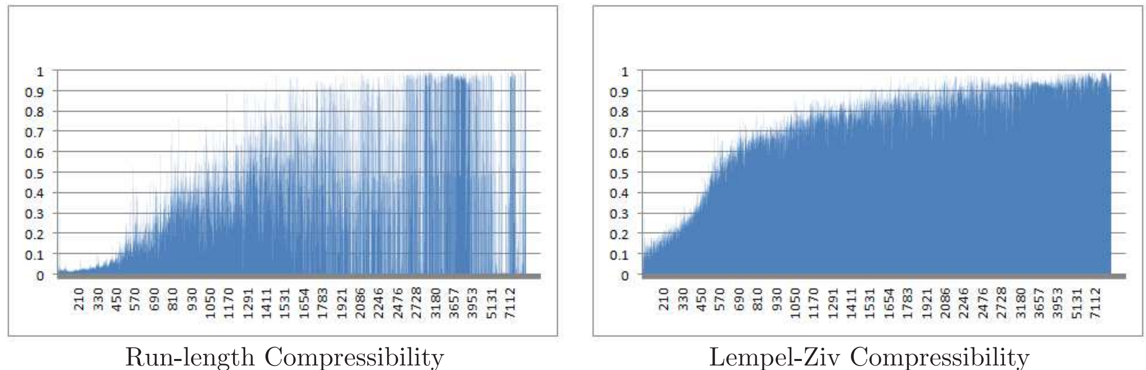
10. Related Work, Variants, and Open Problems

10.1. **Previous Work.** After we developed the main results on local maximum chunking, we learned of a related technique proposed in [26]. That paper introduces the concept of local algorithms for document fingerprinting.



Outlook PowerPoint Virtual Hard disk

FIGURE 6. Distribution of long chunks



Run-length Compressibility Lempel-Ziv Compressibility

FIGURE 7. Compressibility of file data by chunk lengths

In the situation treated in [26], at least one distinguished position, whose file entry (a hash value) is to be part of the fingerprint, must be chosen within each interval of w consecutive positions. The task of choosing these distinguished values is analogous to the task of choosing cutpoints in remote differential compression. In both cases, the chosen positions should be neither too close together nor too far apart, and in both cases agreements between substantial segments of two files should result in corresponding positions being chosen in the two files. Because of the analogy and for the sake of brevity, we shall write "cutpoint" to refer to the distinguished positions, even though [26] does not envision actually cutting files there.

The winnowing method proposed in [26] chooses a cutpoint by considering each interval of hashes with length w and selecting the index with the minimal hash value; if there is a tie, the right-most position with minimal hash value is chosen. As a result, successive cutpoints may be arbitrarily close to each other, but their distance never exceeds w. Winnowing is a local algorithm because whether a position is a cutpoint depends only on hash values within w positions to both sides of it. Analogously to the probability distribution of local maxima, the *density* of local minima is computed as $2/(w+1)$. It is also shown in [26] that any local scheme for choosing distinguished points never farther than w apart must have density at least $\frac{1.5}{1+w}$. It is also shown in [26] that, for the pure point filter method or any impure variant thereof, if the method ensures at least one cutpoint in every interval of length w, then the density of cutpoints is at least $\frac{1+\ln w}{w}$. Finally, [26] gives an algorithm for determining the cut-points.

Our results on local maximum chunking can be viewed as complementary: The winnowing scheme of [26] imposes an absolute upper bound w on the distance between consecutive cutpoints (which we'll call the chunk length) and seeks to prevent the chunks from being too much shorter. Our local maximum method imposes an absolute lower bound $h+1$ on the chunk length and seeks to prevent the chunks from being too much longer. This is why we strive for small slack and for low probability of long chunks.

Finally, we provide an analysis of the average number of file comparisons per file entry, which translates into branch mispredictions.

10.2. **Hash-less Local Maxima.** All of the chunking methods we have considered relied on a pre-processing step that uses a hash function to distill characters in an interval of length w into machine representable numbers. This is useful when a cut-point can be determined by using arithmetic or logical operations on the numbers that are directly supported by the CPU. In the point filter, LBFS, and interval-filter chunking methods, these operations consisted of masking selected bits to determine cut-points. In the local maximum approach, the relevant operation is comparison.

It is however, possible to skip the pre-processing step in the local maximum method and treat a block of w consecutive characters as an $8 \cdot w$ bit number. For numbers of this size, the comparison operation is not directly provided by the CPU, and so we have to supply our own comparison operation to find local maxima among these numbers.

On average, and in practice, it seems that even a naïve procedure for finding locally maximal substrings by means of lexicographic byte-wise comparisons is superior, in terms of running time, to pre-processing $|F|$ with a rolling hash. While a lexicographic byte-wise comparison over words with w bytes requires in the worst case w comparisons per position, it is likely that far fewer comparisons (often just one) are required because the most significant byte of the largest number so far is likely to be larger than a random byte.

One straightforward refinement of the naïve procedure is to record the number of repeated characters at the currently scanned position. This allows processing files consisting of large blocks of the same characters independently of w, but it does not help in the case of files consisting of large blocks of periodic patterns (longer than a single character). Obviously, the first refinement may be generalized to also take periodic patterns into account. In general, it may be of theoretical interest to consider variants of Boyer-Moore [13, 8, 3] string matching algorithms that avoid repetitive scanning of the same characters. In contrast to Knuth-Morris-Pratt and Boyer-Moore string matching algorithms, this problem is not that of finding a fixed pattern, and we cannot rely on a one-pass pre-processing step.

10.3. **Open Problems.** Our calculations leave several problems open:

- What is the slack of the local maximum method? Is it smaller than (our lower bounds for) the slack of the point filter and LBFS chunking methods?
- What is the variance of the chunk length for the local maximum method?
- Replace our estimates for the probabilities of long cutpoint-free intervals by exact values (or tighter estimates).

Another natural question concerns the ratio of expected chunk size to minimum chunk size. We would like this ratio to be as small as possible. The absolute minimum value, 1, is attainable but only by chopping the file into chunks of constant size, and we have seen that this is a bad method because adding one character to a file can destroy all agreement between chunks. The LBFS method can attain ratios arbitrarily close to 1 by taking $h \gg c$ (i.e., $k \gg 1$), but we have seen that this also makes agreement between chunks difficult to attain; recall in particular Remark 50. A reasonable question is how small the ratio can be for local chunking methods. (Locality excludes both constant-length chunks and the LBFS chunking method with $h > 0$.) The interval filter attains a ratio approaching e (for large h), and the local maximum method does somewhat better with a ratio approximately 2. Can a local method do better yet?

Acknowledgements

We are indebted to Dan Teodosiu for initiating and driving the development of the RDC protocols. Joe Porkka developed RDC and its associated protocols that are included in the DFS replication system. Laci Lovász and Yuval Perez helped drawing our attention to ergodic theory. We are also thankful for numerous interactions with Mark Manasse, Akhil Wable and Le Wang on RDC.

References

[1] M. Aigner and G. M. Ziegler. *Proofs from THE BOOK*. Springer Verlag, 2001.

[2] M. Ajtai, R. Burns, R. Fagin, D. D. E. Long, and L. Stockmeyer. Compactly encoding unstructured inputs with differential compression. *Journal of the ACM*, 49(3):318–367, May 2002.

[3] C. Charras and T. Lecroq. *Exact string matching algorithms*. King's College London Publications, 2004.

[4] Microsoft Corporation. Distributed file system replication, 2006. More information on DFSR is available from `http://msdn.microsoft.co/library/en-us/ stgmgmt/fs/distributed _file_system_replication__dfsr_.asp`.

[5] Microsoft Corporation. MSN IM 8 `http://www.msn.com`, 2006.

[6] Microsoft Corporation. Remote differential compression reference, 2006. `http://windowssdk.msdn.microsoft.com/en-us/library/ms715305.aspx`.

[7] Microsoft Corporation. Windows Meeting Space, Windows Vista, 2007. More information is available from `http://www.microsoft.com/windows /products/windowsvista/features /details/meetingspace.mspx`.

[8] M. Crochemore, C. Hancart, and T. Lecroq. *Algorithmique du texte*. Vuibert, 2001.

[9] M. Drinic and D. Kirovski. PPMexe: PPM for Compressing Software. In *IEEE Data Compression Conference*, pages 192–201, 2002.

[10] P. Erdős and G. Szekeres. A combinatorial problem in geometry. *Compos. Math.*, 2:464–470, 1935.

[11] W. Feller. *An Introduction to Probability Theory and Its Applications*. John Wiley & Sons, third edition, 1968.

[12] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley, 1994.

[13] D. Gusfield. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, Cambridge, UK, 1997.

[14] P. Halmos, E. Moise, and G. Piranian. The problem of learning to teach. *Amer. Math. Monthly*, 82:466–476, 1975.

[15] J. W. Hunt and T. G. Szymansky. A fast algorithm for computing longest common subsequences. *Communications of the ACM*, 20(5):350–353, May 1977.

[16] J.W. Hunt and M.D. McIllroy. An algorithm for differential file comparison. Computer Science Technical Report 41, Bell Labs, June 1976.

[17] R. M. Karp and M. O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987.

[18] J. Langford. Multiround RSYNC. Technical report, Dept. of Computer Science, Carnegie-Mellon University, 2001. The report can be obtained from `http://www.cs.cmu.edu/~jcl/research/mrsync/mrsync.ps`.

[19] J. MacDonald. File system support for delta compression. Master's thesis, Department of Electrical Engineering and Computer Science, University of California at Berkeley, 2000.

[20] A. Muthitacharoen, B. Chen, and D. Mazieres. A Low-Bandwidth Network File System. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, pages 174–187, 2001.

[21] C. Percival. Naive differences of executable code. Technical report, Oxford Computing Laboratory, University of Oxford, 2002.

[22] K. Petersen. *Ergodic Theory*. Cambridge University Press, 1983.

[23] M. O. Rabin. Fingerprinting by random polynomials. Technical Report TR-15-81, Harvard University, Center for Research in Computing Technology, 1981.

[24] D. Rasch and R. C. Burns. In-place RSYNC: File synchronization for mobile and wireless devices. In *USENIX Annual Technical Conference, FREENIX Track*, pages 91–100. USENIX, 2003.

[25] M. J. Rochkind. The source code control system. *IEEE Transactions on Software Engineering*, SE-1(4):364–370, December 1975.

[26] S. Schleimer, D. S. Wilkerson, and A. Aiken. Winnowing: Local algorithms for document fingerprinting. In A. Y. Halevy, Z. G. Ives, and A. Doan, editors, *SIGMOD Conference*, pages 76–85. ACM, 2003.

[27] T. Suel and N. Memon. Algorithms for delta compression and remote file synchronization. In Khalid Sayood, editor, *Lossless Compression Handbook*. Academic Press, December 2002.

[28] D. Teodosiu, N. Bjørner, Y. Gurevich, M. Manasse, J. Porkka, and A. Wable. Optimizing File Replication over Limited-Bandwidth Networks using Remote Differential Compression. Technical report, Microsoft Corporation, August 2006.

[29] W. F. Tichy. RCS — a system for version control. *Software — Practice and Experience*, 15(7):637–654, 1985.

[30] D. Trendafilov, N. Memon, and T. Suel. zdelta: An efficient delta compression tool. Technical Report TR-CIS-2002-02, Polytechnic University, Brooklyn NY., June 2002.

[31] A. Tridgell. *Efficient Algorithms for Sorting and Synchronization*. PhD thesis, The Australian National University, February 1999.

Contents

1. Introduction 1
Applicability 5
2. Preliminaries 5
2.1. Files 5
2.2. Ergodic Theory 7

2.3. Useful Formulas 9
2.4. Greedy Increasing Sequences 10
3. Chunking Methods and Slack 12
3.1. Chunking Methods and Locality 12
3.2. Length of Chunks 13
3.3. Costs 16
3.4. Slack from the Left 17
3.5. Slack from the Right 18
3.6. Quality of Chunking Methods 19
4. Point Filter Methods 20
4.1. Pure Point Filters 20
4.2. Point Filters Without Short Chunks 21
4.3. The Slack of LBFS Chunking 24
4.4. The Reverse Slack of LBFS Chunking 30
5. Interval Filter Methods 31
5.1. Definition of Interval Filter Chunking 31
5.2. Statistics of Interval Filter Chunking 31
5.3. Slack of the Interval Filter Method 33
5.4. Probability of Long Chunks 35
5.5. Reverse Slack of the Interval Filter Method 39
6. Local Maximum Chunking 39
6.1. Statistics of Local Maximum Chunking 40
7. Probability of Long Chunks 41
7.1. Pure Point Filter Method 42
7.2. LBFS Method 42
7.3. Interval Filter Method 43
7.4. Local Maximum Method 43
8. Computing Local Maxima 47
9. Evaluation and Experiments 52
9.1. Computation overhead 53
9.2. Intra-file Compression 53
9.3. Distribution of Long Chunks 54
9.4. Run-length compression 54
10. Related Work, Variants, and Open Problems 55
10.1. Previous Work 55
10.2. Hash-less Local Maxima 57
10.3. Open Problems 57
Acknowledgements 57
References 58

Microsoft Research, One Microsoft Way, Redmond, WA 98052, U.S.A.
E-mail address: nbjorner@microsoft.com

Mathematics Department, University of Michigan, Ann Arbor, MI 48109–1043, U.S.A.
E-mail address: ablass@umich.edu

Microsoft Research, One Microsoft Way, Redmond, WA 98052, U.S.A.
E-mail address: gurevich@microsoft.com