
# Creating a standardized corpus for digital stratigraphic methods with fsstratify

By:
Julian Uthoff, Lisa Marie Dreier, Martin Lambertz, Mariia Rybalka, Felix Freiling

From the proceedings of
The Digital Forensic Research Conference
**DFRWS APAC 2025**
Nov 10-12, 2025

Forensic Science International: Digital Investigation 54 (2025) 301986

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

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

DFRWS APAC 2025 - Selected Papers from the 5th Annual Digital Forensics Research Conference APAC

# Creating a standardized corpus for digital stratigraphic methods with fsstratify

Julian Uthoff [a,*], Lisa Marie Dreier [b], Martin Lambertz [c], Mariia Rybalka [c], Felix Freiling [b]

[a] Hamburg State Police, Hamburg, Germany
[b] Computer Science Department, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Erlangen, Germany
[c] Fraunhofer FKIE, Bonn, Germany

ARTICLE INFO

*Keywords:*
Digital Stratigraphy
File Systems
Dataset Creation
Digital Forensics

ABSTRACT

Digital stratigraphic methods aim to infer new information about digital objects using their depositional context. Many such methods have been developed, for example, to interpret file allocation traces and thereby estimate timestamps of file fragments based on their position on disk. Such methods are difficult to compare. We therefore present a corpus of NTFS file system images that can be used to evaluate these methods. The corpus comprises different categories, each extensively employing a small subset of file system operations to display their effect on file allocation traces. We demonstrate the usefulness of this corpus by evaluating the method of Bahjat and Jones (2019) that derives the timestamp of a file fragment from the timestamps of neighboring files. The corpus was generated using a revised version of `fsstratify`, a software framework to simulate file system usage. The tool is able to log the position of content data during file creation, greatly facilitating research in the realm of digital stratigraphy.

## 1. Introduction

Timestamps stored on digital media are an essential category of evidence used ubiquitously in digital forensic investigations. Using timestamps, for example, to create digital forensic timelines, is, however, often a cumbersome undertaking (Metz, 2021) because of clock skews, timezones and other context specific rules for their creation (Chow et al., 2007). While many approaches tackle these issues, any technique that relies on timestamps is bound to fail if no timestamps are available. Such situations can occur if files have been deleted and their file system metadata entries (including timestamps) have been overwritten with other data. For example, consider a case where investigators find several illegal but deleted files on the device of a suspect *A* and *A* claims that these files originated from a different user *B* who used the device before *A* started using it. Indeed, it would be helpful for the investigation if analysts could determine the creation times of these files. But how can this be done if no timestamps are available?

In such cases, digital forensic analysis can build upon methods developed in other fields like archeology that also focus on questions of chronology but where timestamps are not as readily available as in file systems. In these areas, chronological information is inferred from the *placement* of objects relative to other objects. For example, in *archeological stratigraphy* (Harris, 1989), objects found in a discernible excavation layer (stratum) can be dated in relation to other strata, the common case being that lower strata are older. These insights have inspired the digital forensics community to study the implications of *file allocation traces*: They can provide insight into "origin, composition, distribution, and time frame of strata within storage media" (Casey, 2018), giving rise to the field of *digital stratigraphy*. In this paper, we focus on digital stratigraphic methods for *chronology*, i.e., the chronological dating of file system data in contexts where critical timestamp data is either unavailable or not trustworthy. These provide, for a given digital object *f*, an estimation of the time when *f* was created, used or deleted. Ideally, aside from the time estimation, such methods also yield an estimation of reliability like a margin of error or error probability.

* Corresponding author.
*E-mail addresses:* julian.uthoff@mailbox.org (J. Uthoff), lisa.dreier@fau.de (L.M. Dreier), martin.lambertz@fkie.fraunhofer.de (M. Lambertz), mariia.rybalka@fkie.fraunhofer.de (M. Rybalka), felix.freiling@fau.de (F. Freiling).

https://doi.org/10.1016/j.fsidi.2025.301986

Available online 3 November 2025

### 1.1. Related work

Casey (2018) coined the term *digital stratigraphy* for digital forensic methods that take placement information into account in 2018. But placement information of data units especially from deleted files has been used for multiple purposes before. For example, Garfinkel et al. (2010) used this data as one source of information to determine the user that most probably created and used the file before deleting it. Several such methods have been proposed for chronological dating. Most of them, however, lack proper validation and reliability estimation. A prominent file fragment dating method was designed for NTFS (Bahjat and Jones, 2019) and FAT (Bahjat and Jones, 2023), but it was evaluated on disk images without a known ground truth.

Another method was proposed by Willassen (2008). He tried to infer the creation order of files by three properties: by their placement in the MFT, by an attribute counting how often the MFT entry was reused and by sequence numbers in order to find timestamps that were antedated. The approach's evaluation was based on a small study, where four subjects with different technical skills had to antedate a file. While the description of the antedating process yields some knowledge about temporal relationships, it does not provide a real ground truth. In fact, Palmbach and Breitinger (2020) tried to re-implement this method but failed to confirm prior observations on a pre-determined scenario.

Other work has focused more on the placement strategies of file systems rather than chronological dating. Karresand (2023), for example, ran several virtual machines on a cluster and logged the $Bitmap file after each file system operation to understand how these operations (statistically) affect the placement of files in the file system. A greater emphasis on the temporal behavior of cluster allocation in file system drivers was placed by Schneider et al. (2024) who built the *File System Activity Simulator* (FSAS) (Schneider et al., 2024), a system that monitors file system activity at the driver level. A similar tool called `fsstratify` was developed independently by Bojic et al. (2020). In the present work, we extend and use `fsstratify` as FSAS was not yet available at the onset of our research.

### 1.2. Contributions

The lack of proper evaluation can be regarded as the Achilles' heel of many digital forensic methods (Horsman, 2019). To alleviate this situation in the field of digital stratigraphy, we contribute a dataset for chronological dating. More specifically, we provide a corpus of NTFS file system images which comes with detailed information about the ground truth, i.e., which file system operations happened when and in which way, such that the chronological circumstances of every bit on each image is known. The corpus is *not* intended to provide examples of representative file system usage. Rather our corpus, like other datasets for tool testing (Nemetz et al., 2018), can be used to benchmark and thereby compare different dating methods and tools.

We demonstrate the usefulness of the corpus by evaluating the dating method of Bahjat and Jones (2019) that attempts to estimate the creation and deletion times of file fragments found in slack space from timestamps of neighboring files. These two estimations yield an approximation of a file fragment's active lifetime, bounding it from below (lower bound, creation time) and above (upper bound, deletion time). To conduct a differentiated evaluation, we defined the measures of *precision* and *accuracy* as favorable properties in Section 4.1: While *accuracy* is the probability that the estimated value is within a certain bound of the true value, *precision* is the probability that the lower bound is in fact below the true creation time and that the upper bound is in fact above the true deletion time as defined. We show that, in general, the *accuracy* of Bahjat and Jones (2019) is rather low for both, lower and upper bound, on the data in our corpus and that the accuracy can be improved by a few minor adaptions. However, the precision of Bahjat and Jones (2019) is very good, in particular with regard to the upper bound. None of these insights could have been achieved without a dataset that has ground truth.

As an additional contribution, we take and modify `fsstratify`, a tool by Bojic et al. (2020), that is able to collect stratigraphic data about the placement of blocks on disk during file write and delete operations. The modified version of `fsstratify` is more robust than the previously published version and was extended by the notion of *data generators* that allow to define the content of files written. The updated version also now allows to control the time in which file system operations happen.

The corpus with additional supplemental material (Uthoff et al., 2025), the evaluation scripts (Dreier, 2025) as well as the code of `fsstratify` (Lambertz and Rybalka, 2025) are available online.

### 1.3. Outline

This paper is structured as follows: We first give some background on `fsstratify` and the modifications we made to the tool. We then introduce the corpus and the rationale behind it in Section 3. In Section 4 we use the corpus to evaluate the method of Bahjat and Jones (2019) for file fragment dating. We discuss our results in Section 5 and conclude in Section 6.

## 2. Revising `fsstratify`

The idea of `fsstratify` (Bojic et al., 2020) is to execute different file operations and to record the resulting changes in the file system in a log file. The framework performs the operations using the file system implementation of the operating system it runs on. By using the same simulation scripts on different operating systems, we can evaluate and compare the behavior of their file system implementations and peculiarities introduced by the operating systems. Tracking changes of the on-disk representation of the file system allows detailed analyses of the allocation strategies and the behavior of metadata properties such as timestamps. This makes `fsstratify` particularly useful for digital stratigraphy, as presented in this paper, and also for generating data sets for file carving evaluations.

Fig. 1 illustrates the functionality of the framework on a conceptual level. `fsstratify` performs predetermined actions (file operations) on mounted file systems to create aged file systems. The sequence of file operations to execute is specified by so-called *playbooks*. A playbook is essentially a script with one operation per line. Playbooks can be written manually or generated on the fly based on *usage models*. After every executed action, i.e., after every file system operation, the *file system analyzer* parses the file system and logs any changes to the state of the file system. The changes logged include information about the currently used clusters and metadata about the files, such as timestamps.

The following sections provide more details on each component, highlighting important aspects and design decisions. Note that the framework is highly modular, and almost all components can be

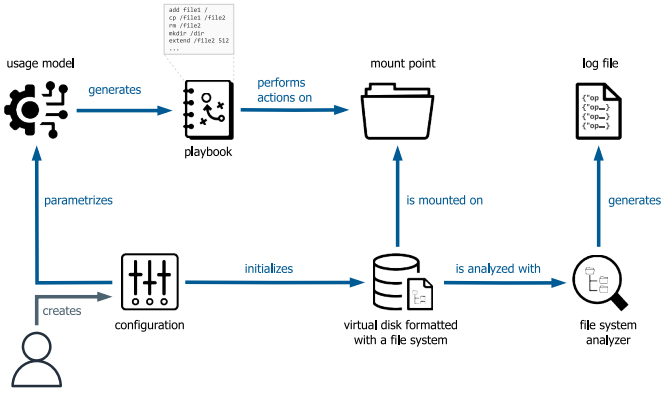



**Fig. 1.** Conceptual overview of `fsstratify`.

changed, extended, or configured.

### 2.1. Simulation volumes and file system analyzer

`fsstratify` uses a simulation volume to format a file system on. The framework supports different volume types for its simulations. The simplest type is the file-based volume, where a file in the host file system is used, but simulations on actual partitions or hardware disks are also supported. While the former is more convenient, the latter commonly enables larger volume sizes.

In a typical simulation, the volume is initialized before executing any operations. This initialization involves formatting the volume with the desired file system using the specified parameters, ensuring a clean and consistent file system state at the start of the simulation. However, there are scenarios where using a pre-existing, non-pristine (or "dirty") volume is preferable, such as when the outcome of one simulation is intended to serve as the starting point for subsequent simulations. To support such use cases, `fsstratify` allows simulations to utilize existing dirty volumes and retain volumes after a simulation is completed.

Parsing the on-disk structures of a file system to obtain information about allocated clusters and file metadata is a complex task. In its initial version, `fsstratify` used the popular analysis software The Sleuth Kit (TSK)[1] for this purpose. Because TSK's NTFS parser repeatedly crashed simulations, we replaced TSK with the tool *dissect*.[2] This framework is written purely in Python, greatly enabling its cross-platform portability.

### 2.2. Playbooks and operations

Playbooks define the operations to be carried out on a mounted file system. The operation syntax is similar to well-known command line utilities, easing the reading and writing of playbooks. Fig. 2 shows a typical execution of a playbook where playbook commands like `write` and `rm` are given on the left. On the right, typical results of the playbook commands are depicted, i.e., sequences of data units that were allocated or deallocated during that step. These are extracted by the file system analyzer and logged to a file. The projection of all data units to the most recently recorded change for a step results in the *stratum* of that step.

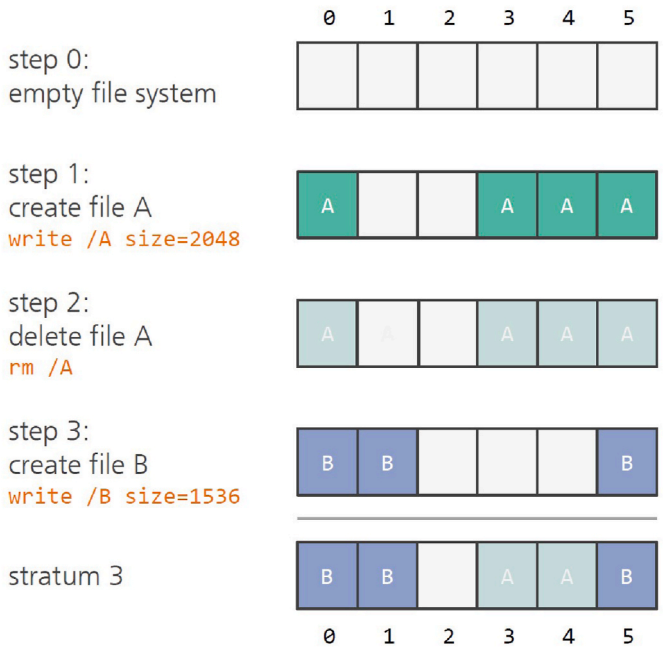



**Fig. 2.** Playbook commands (left) and views of the changes in the allocated data units after executing the commands with `fsstratify` (right). The projection of all blocks to the latest recorded changes for a step yields the stratum of the final step.

The playbook commands are implemented using functionality provided by the Python standard library, which, in turn, uses the operating system interfaces. This approach ensures that the behavior of the file system driver of the operating system under test is reflected in the simulation results. This also maintains the artifacts generated by other parts of the operating system like setting timestamps.

In the original `fsstratify`, data-writing operations like `write` and `extend` used randomly generated data, which benchmarking revealed as the primary performance bottleneck. To mitigate this, we introduced *data generators*. These generators let users define the type of data to write during simulations. To ensure backward compatibility, we implemented a random data generator reflecting the original `fsstratify` behavior. Additionally, we implemented a generator writing an easy-to-identify pattern. The pattern consists of the file name, an incrementing number for each 512-byte chunk of a file, and a static byte filling the remainder of the fragment. We used this pattern-based generator for our data set, as it enables effective downstream compression of disk images and simplifies manual analysis of the raw data.

The operations available in the initial `fsstratify` version only employed file-modifying functionality and did not interact with the operating system in any other way. While this eases running simulations on regular systems and avoids the necessity of dedicated simulation systems, we found that the operations were carried out in too close succession. We wanted to be able to control the temporal difference between two file operations or, more precisely, the time-related metadata caused by the operations. We implemented this as an additional `sleep` command which waits for a desired time before executing the next operation. While having no side effects, using this approach dramatically slows down the simulations. As such, we implemented an additional `time` command which sets the system time of the operating system running the simulation to a specific value. This command may cause side effects within the operating system (Vanini et al., 2024), so it must be used carefully. Still, we used it in our experiments since the simulation volumes do not contain operating system files or files written by other programs.

### 2.3. Differential logs

`fsstratify` records the changes to the file system after every simulation step in a machine-readable format. Log lines contain the information necessary to infer the state of the file system at any given step during the simulation. The recorded information includes the executed operation with its parameters and the files and directories it affects. An operation affects a file when its allocated data units or its metadata changes. Metadata changes include not only facets like updated timestamps but also changed paths. When a directory is moved, for example, all files in this directory will be affected. The exact behavior can be configured with simulation parameters.

For each of the affected files their currently allocated data units are logged. Moreover, `fsstratify` includes various file system metadata about the files, such as MACE timestamps. These are file system timestamps that save each file *M*odification, *A*ccess, *C*hange and metadata *E*ntry change. All metadata logged depends on the file system under test, as not all file systems use the same set of metadata.

## 3. Introducing the corpus

We now present the forensic corpus for temporal analysis. It is a

[1] https://www.sleuthkit.org/sleuthkit/.
[2] https://github.com/fox-it/dissect.

corpus of 74 disk images, each containing a single NTFS file system, structured into 12 *categories* that resemble typical problem scenarios for digital stratigraphy. The objective is *not* to create data that is in any way realistic or representative of real-world usage but rather to create test data sets for which a ground truth of data creation exists. Therefore, the file content contains byte values devoid of any intrinsic meaning.[3] No other file types like video or audio formats were employed when writing to the disk images. As a result, the files do not contain any information that could be considered privacy relevant and would need to be anonymized.

### 3.1. Creating the corpus

The disk images vary in size (500 MB, 5 GB and 50 GB) and were created using two different operating systems (Windows 11 and Ubuntu 24.04). Files and directories were created and populated in random fashion, with file sizes varying between approximately 2000 bytes to over one gigabyte. The results therefore reflect the behavior of the respective file system driver during various file operations and across different disk sizes. We generally used the default settings but disabled the trim command as trim wipes the slack space of files, which zeroes all fragments. For some of the image files in the corpus, a longer period of utilization was simulated by adjusting the system time. The file operations were conducted over a simulated period of up to three years.

We structured our corpus into one directory per category, each containing one directory per image. Still, the corpus contains additional information for each image: the Simulation.playbook (containing the fsstratify operations) and the Simulation.yml (containing the configuration of fsstratify). Both are needed to recreate the corpus. Additionally, we added the *Simulation.strata*, which is the log fsstratify creates during running the scenario. It contains the ground truth, e.g., creation timestamps and allocated blocks for each operation.

### 3.2. Corpus categories

As mentioned above, the corpus is structured into a total of 12 *categories* that resemble typical problem scenarios for digital stratigraphic methods, i.e., circumstances where files have been deleted, expanded, copied, moved or overwritten. Note that, since all simulations were performed on a single file system, file moves are equivalent to renames and do not involve copying the original data. The following briefly describes the circumstances and rationale of each scenario.

*Category 01: File creation until disk space is used up.*

In the first scenario, files were written until the disk was filled to capacity. This setup is intended to demonstrate how the NTFS driver manages write operations under disk saturation conditions. As the allocation behavior of some file system drivers change in such a context, this category ensures digital stratigraphic methods can deal with it.

*Category 02: Deleted files.*

In the second scenario, the file systems were initially populated with a variety of files through the use of the write operation. Subsequently, files were deleted at different locations within the file system. In most cases, files were solely marked as deleted, but not overwritten, so the file content can still be located within the file system. Such image files can be employed to ascertain the date of deleted files.

*Category 03: Write Operation - Deleted file fragments.*

The third scenario extends the procedure described in the second scenario. The contents of the deleted files were overwritten with new data, rendering only fragments of the deleted file contents recoverable. Due to the overwriting process, the deleted file contents are no longer located solely in the unallocated file system area; fragments are now also within the slack space of newly written files. This scenario permits the dating of deleted files in both the unallocated regions of the file system and in the file slack of other files. Given that only the write operation was utilized to create files in this instance, it is possible to conduct a detailed examination of the specific effects of this operation.

*Category 04: Copy Operation - Deleted file fragments.*

In the fourth scenario, an initial set of files was created using the write operation before a large number of files were generated by copying existing ones. Additionally, many of these files were subsequently deleted and overwritten, producing fragments of deleted files. Since the access timestamp of the source file is updated during copying (Chow et al., 2007), the resulting timestamps of the deleted file fragments may differ from those observed in the third scenario.

*Category 05: Move Operation - Deleted file fragments.*

The idea of this scenario was consistent with that of the fourth scenario, except that most existing files were relocated to varying directories using the move operation instead of copying them. As in the preceding category, the goal is to ensure that a large number of files in the file system are affected by the move operation, thereby testing the effects of this operation.

*Category 06: Extend Operation - Deleted file fragments.*

In the sixth category, the strategy focuses on the extend operation: The strategy of creating a group of files and then extending each of them was repeated multiple times, with specific files being deleted at designated positions. This category distinguishes from the others regarding timestamps as the extend operation results in a comprehensive update of all timestamps except the creation timestamp. Furthermore, the alternating recreation and extension of files results in a higher degree of file fragmentation than that observed in the third scenario.

*Category 07: Overwrite by write operation.*

This scenario addresses the process of overwriting existing files. Therefore an initial group of files was created. But rather than deleting existing files, new data was stored under file names that already existed in the file system, thereby overwriting existing files. It is usually expected that the timestamps are updated, except for the creation time. When overwriting, we ensured that not all of the file content was always overwritten, resulting in the persistence of fragments of the previous file within the file system. We overwrote files in disparate locations within the file system to evaluate the behavior of file dating methods at varying positions within the file system.

*Category 08: Overwrite by move operation.*

As in the seventh scenario, existing files in the file system were overwritten, creating unallocated file fragments. However, no new files were created while overwriting; instead, the contents of existing files from the file system were moved to an existing destination file. Again, we expect that the timestamps are updated, except for the creation timestamp. The behavior during cluster allocation may differ from that observed in the seventh scenario, which could result in disparate outcomes when calculating the creation dates of deleted file fragments.

*Category 09: Overwrite by copy operation.*

As in the seventh and eighth scenarios, the overwriting of existing files resulted in the generation of unallocated file fragments within this category. However, the overwriting was conducted by copying existing file contents to a destination file with a file name already existing. Since we are not aware of any systematic research that has studied this situation, we added several images to the corpus as additional challenges for chronological dating of unallocated fragments.

*Category 10: Fragmented files.*

This scenario focuses on creating file systems with a high degree of fragmentation. If the content of files is stored in multiple fragments, the file system will automatically show a greater interlacing of files. Dating methods, such as that of Bahjat and Jones (2019), calculate creation dates based on the creation timestamps of neighboring files. A greater degree of interlacing is likely to produce different results than with non-fragmented files. To achieve a high degree of fragmentation, we have pursued the following strategy in this category: First, the disk was filled to a high degree with files using write commands. Afterwards, a

[3] To facilitate analysis, the files still contain the file name at regular intervals, though.

group of operations was repeated: deleting two non-adjacent files and writing a new file, which is either slightly smaller or larger (if space allows). This way, every new file is at least fragmented into two fragments as not enough space is available to write it consecutively.

*Category 11: Manipulated timestamps.*

Assume we can determine the temporal relationships between files based on their positions in the file system. In that case, it should also be possible to identify manipulated timestamps by examining the position of a file. As Casey (2018) notes, stratigraphic analysis can be a valuable tool for detecting timestamp manipulation. While there is currently no research in this area, image files have been created to test this hypothesis. In these corpus files, the timestamps in the metadata were manually manipulated for several files at different positions in the file system. While we could have chosen any category as basis for the manipulation, we've decided to build on the strategy of the first category as it the simplest.

*Category 12: Manipulated timestamps in fragmented files.*

In the final scenario, the timestamps of files were again altered at various points within the file system. Compared to the previous category, the file system was highly fragmented before the manipulations, analogous to the tenth scenario. It is presumed that identifying altered timestamps of fragmented files is more challenging than in the eleventh category due to the interlacing of files within the file system.

## 4. Leveraging the corpus for digital stratigraphic analyses

In this section, we use our corpus to re-evaluate the file fragment dating method proposed by Bahjat and Jones (2019). Originally, Bahjat and Jones evaluated their method on the "M57 Patent digital corpora built by the Naval Postgraduate School" with reference to Garfinkel et al. (2009), considering only a subset of the images, namely "Pat's drive." This corpus does not provide a ground truth, but Bahjat and Jones try to estimate the ground truth from the 17 snapshots of Pat's drive over time. Still, we argue that our corpus can improve their evaluation because having a known ground truth improves the evaluation's reliability.

We include various variants of the method of Bahjat and Jones and also classical estimation approaches in Section 4.3 to show that our corpus is valuable for comparing different methods. Furthermore, we refined the evaluation methodology by introducing accuracy and precision metrics and by distinguishing lifetime estimation and lifetime bounding.

### 4.1. File fragment dating

File fragment dating is typically used when a fragment of file $f$ is found in the slack space of another (active) file $\omega$ and the investigator needs to assess the "lifetime" of the original file $f$, meaning the true value $\tau$ of the creation time of $f$ and its deletion time $\delta$. There are two variants of the file fragment dating problem.

**Definition 1.** **(Lifetime estimation)** Estimate the lifetime of $f$ as precisely as possible, i.e., give estimations $c$ of creation time and $d$ of deletion time of $f$ such that $|c - \tau| < \epsilon$ and $|d - \delta| < \epsilon$ for small $\epsilon$.

**Definition 2.** **(Lifetime bounding)** Estimate bounds on the lifetime of $f$ as precisely as possible, i.e., determine values $c$ and $d$ such that $c \leq \tau$ and $\delta \leq d$ and that $\tau - c$ and $d - \delta$ are minimal.

Note that lifetime bounding adds an additional constraint to lifetime estimation, namely that both estimations need to be strict bounds, i.e., the estimated creation time $c$ must be below the true creation time $\tau$ and the estimated deletion time $d$ must be after the true deletion time $\delta$.

As both types of file fragment dating have different goals, they need to be evaluated with different metrics. For this, we define the following three metrics.

**Definition 3.** **(Accuracy)** The accuracy of the calculated value is its absolute difference from the true value.

**Definition 4.** **(Precision of lower/upper bound)** An estimated lower bound is precise iff it is below the true value. An estimated upper bound is precise iff it is above the true value.

### 4.2. Bahjat and Jones explained

Bahjat and Jones (2019) performed lifetime bounding, meaning they estimated a lower and an upper bound for the creation and deletion times of a file fragment. They roughly outlined their method in their paper, but they did not provide an actual implementation. For that reason, we re-implemented it based on our interpretation of the method as outlined in the following.

Bahjat and Jones define the *slack owner* as "the file occupying the first sector of the evidence file". We interpret this as the active file, that has the file fragment in the slack space of its (last) cluster. Based on the concept of slack owner, Bahjat and Jones (2019) calculated the upper bound (of the deletion time) as the "maximum of the eight dates found in the Slack-Owner [*sic*]". Our interpretation of this method is given as pseudo code in Fig. 3.

Even though not directly explained in the paper, we assume that the reason for taking the maximum of *all* timestamps instead of the maximum of the two creation timestamps in the Filename and Standard Information Attribute lies in the following fact: some file operations (e.g., copy) change the file content's location but not the creation timestamps under certain circumstances, e.g. when file tunneling is triggered or when a file is copied to an already existing file name and overwriting it (Bouma, Jonker, van der Meer and Aker, 2023). Thus, taking the maximum of all timestamps should give a highly precise albeit not necessarily particularly accurate upper bound.

To calculate the lower bound, two further concepts need to be introduced: the *estimated creation time* and the *k nearest neighbors*. Firstly, the estimated creation time (also called "TrueDate" and sometimes "TrueCreate" by Bahjat and Jones) is used as the ground truth in their evaluation as well as for calculating the lower bound. They distinguished between a file created on the disk and a file transferred from outside the disk and took either the file's creation date or its modification date respectively as the estimate. Fig. 4 shows our interpretation of the algorithm in pseudo code. Since we have a ground truth available in our corpus, we use this approach to calculate the lower bound in the evaluation.[4]

Secondly, the most central concept used by Bahjat and Jones (and for some variants introduced in Section 4.3) is determining the $k$ nearest neighbors. Our interpretation of the algorithm doing so is shown in Fig. 6. There, Bahjat and Jones searched for the $k$ nearest (in regards to disk position) files with the following properties:

- At least one of their data runs precedes the fragment on disk.
- Their estimated creation time is earlier than the "upper-bound date" of the fragment.

```
1  calc_upper_bound(fragment fr):
2    sl_owner = get_slack_owner(fr)
3    max_date_std = max(sl_owner.std_info.mace)
4    max_date_fn = max(sl_owner.file_name.mace)
5    return max(max_date_std, max_date_fn)
```

**Fig. 3.** Pseudo code of *calc_upper_bound*

[4] According to Bahjat and Jones the conditional statement in Fig. 4 should only be relevant for files coming from an external disk. Rather unexpectedly, we found files that triggered this condition in our corpus although we do not have a category that copies files between volumes. Upon closer examination, this condition was true only for exactly two files that seem to be related to the process of adding the disk as external disk in the simulation.

```
estimate_creation(file f):
  if f.file_name.c_date > f.std_info.e_date:
    estimation = f.file_name.c_date
  else:
    estimation = f.file_name.m_date
  return estimation
```

**Fig. 4.** Pseudo code of *estimate_creation*

```
calculate_lower_bound(fragment fr):
  neigh_creations = nearest_neighbors(10, fr)
  avg_creation = (sum(neigh_creations) /
                  count(neigh_creations))
  return avg_creation
```

**Fig. 5.** Pseudo code of *calculate_lower_bound*

```
nearest_neighbors(int k, fragment frag):
  sl_owner = get_slack_owner(frag)

  # List all dataruns preceding frag
  neighbor_start_clus = []
  for each active_file in file system:
    if active_file == sl_owner:
      continue
    for each r in active_file.dataruns:
      if r.start < sl_owner.last_datarun.start:
        neighbors_start_clus.append(r.start)
  sort(neighbors_start_clus, order=descending)

  # Take the k nearest/preceding data runs of
  # different files with their estimated
  # creation time being earlier than the
  # upper bound of the fragment
  neighbor_creations = []
  for each x in neighbors_start_clus:
    if length(neighbor_creations) == k:
      break
    if x.file is a file in neighbor_creations:
      continue
    est_creation = estimate_creation(x.file)
    if est_creation > calc_upper_bound(frag):
      continue
    else:
      neighbor_creations.append(est_creation)
  return neighbor_creations
```

**Fig. 6.** Pseudo code of *nearest_neighbors*

While not directly explained in the paper, we understand the rationale behind this choice of properties as follows. First, NTFS drivers mostly fill a disk from lower sectors towards higher ones. Hence, limiting the neighbors to the preceding ones should generate a better precision of the lower bound as the files should typically have been created *before* the file fragment. Still, using the disk from lower to higher sectors is only a general rule with file deletions causing many exceptions: When a file is deleted, the clusters containing its content are reused at some point, often creating areas where position and creation time do not directly correlate to their neighbors. The second condition attempts to detect this and therefore neglects neighbors that have a creation time later than the upper bound. This ultimately means that they were created after the original file was deleted, and therefore (by definition of the upper bound) *after* the slack owner was created.

Bahjat and Jones finally sought to determine the lower bound as follows (also shown in Fig. 5): They calculate the average of the estimated creation time of the 10 nearest neighbors. They picked $k = 10$ because they found that "10 neighbors are sufficient to calculate the average and the average does not drop significantly by adding more nodes"(Bahjat and Jones, 2019).

### 4.3. Variants and alternative approaches to file fragment dating

To show that our corpus is not adjusted for one particular method, we decided to additionally evaluate some variants of the bound estimations presented in Section 4.2. We took a naïve approach to an alternative upper bound and varied the estimation approach built upon the $k$ nearest neighbors for the lower bound.

Overall, we considered two approaches to estimating the *deletion time* of the fragment:

- **Bahjat and Jones' upper bound (BJ)** as described in Section 4.2.
- **naive upper bound (naïve)**: Instead of taking the maximum of all 8 timestamps of the slack owner, we only use the maximum of the creation timestamps in the $STANDARD_INFORMATION attribute and the $FILE_NAME attribute. That way, the precision will likely be somewhat lower as factors like file tunneling may maintain a creation time which is "too early" for the position, but the bound still could yield a higher accuracy if the majority of operations changing the position of the file content also changes the creation timestamps.

We also considered the following five variants and classical estimation approaches to estimating the *creation time* of the fragment:

- **average (by Bahjat and Jones) (avg)** as described in Section 4.2.
- **median (med)** is an adaption of the method by Bahjat and Jones where the *median* of the creation times of the $k$ nearest neighbors is used instead of the average. In this way we hope to achieve an increase in precision as the median is mathematically less affected by outliers.
- **minimum (min)** is a similar adaption of the method of Bahjat and Jones using the *minimum* of the $k$ nearest neighbor creation times instead of the average. The idea of this method is that an outlier with a timestamp later than the creation time can only influence our estimation if no earlier neighbor exists. This should be highly precise but we expect to have a significantly lower accuracy than for the other methods as the oldest neighbor is not necessarily a good estimation of the fragment's creation time. For this reason, this method might be better suitable for lifetime bounding than for lifetime estimation.
- **linear regression (linReg)** uses a simple linear model to project the slope of the rising flank of creation times to predict the creation time of the slack owner. The method approximates a line by fitting it to

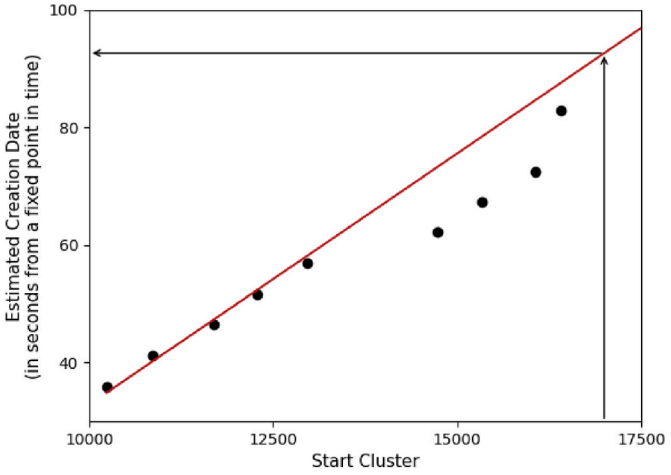



**Fig. 7.** Linear Regression between start cluster and estimated creation time: A linear model (red line) is approximated from the 10 nearest neighbors (points), through which the creation time can be estimated based on its start cluster (arrows).

the disk positions and the estimated creation timeof the $k$ nearest neighbors as shown in Fig. 7.

- **linear regression within interquantile range (linRegQuant)** is an adaption of linReg that attempts to reduce the effects of outliers by trying to exclude them: Instead of considering all 10 nearest neighbors, we only take those in the interquartile range, i.e., 25 % of the data above and below the median.

### 4.4. Results

This section presents the results from the evaluation of different file fragment approaches described above using our corpus. Our evaluation only uses images from the corpus that contain file fragments, specifically those in categories 03 to 10 created on Windows. The images created on Ubuntu did not create fragments in the slack space, instead the slack space was zeroed out. As discussed in Section 4.1, we use precision and accuracy as metrics to assess how suitable each method is for lifetime estimation and lifetime bounding.

#### 4.4.1. Evaluating the upper bound

Fig. 8 shows the results for the accuracy of the upper bound using BJ and naïve. The horizontal axis plots the accuracy interval $\epsilon$ as the percentage of the disk runtime within which the estimated bound is considered accurate. This should lead to a monotonic increase in the measurements as a larger interval can only cause more fragments to be dated accurately. As shown in Fig. 8, the graph for BJ increases much faster than that of naïve, so its accuracy is clearly better.

In comparison to the original paper, the accuracy of BJ increases faster: In the original paper, the accuracy is over 80 % based on a interval of ±5 days around the correct date being considered "accurate" (Bahjat and Jones, 2019). With the scenario having a runtime of 33 days, this equates to around ±15 % runtime, whereas the graph is above 90 % accuracy within a deviation interval of ±15 % runtime. The reason for this appears to stem from the way in which we created the corpus since we focused on one file creation method per category.

When calculating the precision of BJ, we found that it is precise in almost 100 % of all cases. The precision of naïve, however, is much more diverse: it ranges from 9.5 % to 80 %, depending on file or category. This makes the BJ the better choice for lifetime estimation as well as lifetime bounding.

#### 4.4.2. Precision of the lower bound

We now evaluate the precision of the lower bound. The results are depicted in Fig. 9. The horizontal axis shows the number of neighbors $k$ with respect to the percentage of estimations that are precise. Here, min clearly outperforms the other algorithms in terms of precision. This is not unexpected: A fragment is precisely bounded if—among all its nearest neighbors—it contains at least one neighbor that was created before the fragment, which is the case for a majority of fragments. It even performs so well that we decided to look at the raw data. Here, it is evident that the method (when considering at least 20 nearest neighbors) has a precision of 100 % for all categories but category 10, which was specifically designed to contain a huge amount of fragments. This shows that our corpus was designed in a way so that it covers relevant edge cases.[5]

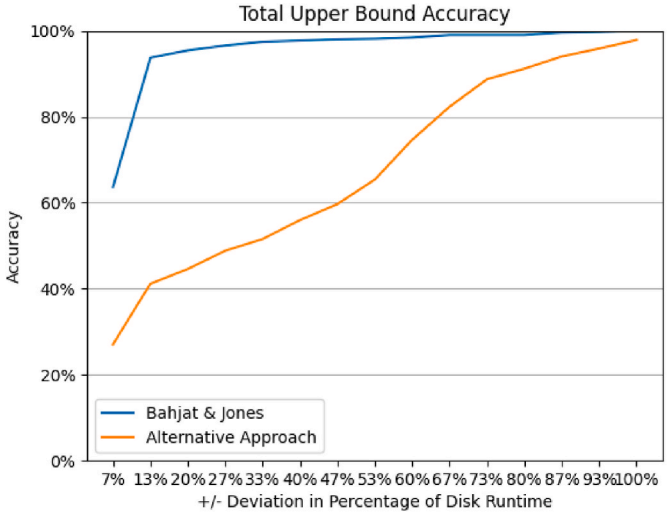



**Fig. 8.** Accuracies of the upper bounds.

Overall, no other approach comes close to the precision of min, even though med has a high precision as well. It is followed by linRegQuant and avg, with their order depending on the question of which upper bound is used to restrict outliers if any. This is because avg and linReg profit from using any upper bound (especially naïve) to restrict outliers as the outliers' estimated creation time is higher than the upper bound. While every approach at least minimally profits from using naïve as an upper bound, avg and linReg appear to have been greatly influenced. avg increases its precision by more than 20 % and linReg, which showed the worst performance for precision, started to show a trend: while it was only fluctuating when adding more neighbors without any upper bound, it started to show a maximum around 10 %–20 % and stayed equal at a high number of 50–100 neighbors. This is untypical as an increase is the expected behavior for a good file fragment dating method: Adding more neighbors should primarily add more *older* neighbors, which decrease the estimated date and thus increases the precision of the lower bound. Not having such an increase is a sign that the influence of outliers outweighs the influence of the neighbors created before the fragment.

#### 4.4.3. Accuracy of the lower bound

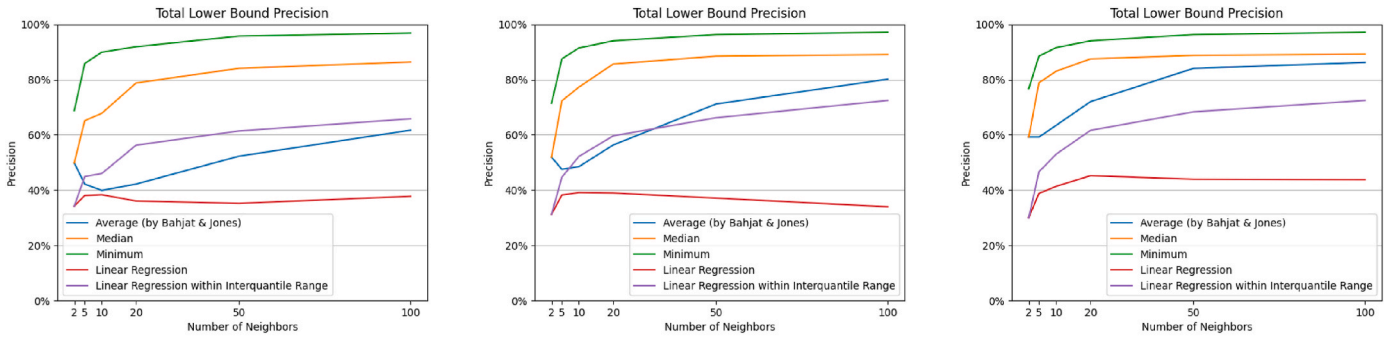
Fig. 10 gives an overview of the results for the accuracy of the lower bound using 20 nearest neighbors for different methods. Here, min clearly performs worst, which was expected: Taking the minimum of the $k$ nearest neighbors typically underestimates the true creation time. This gives a high precision as seen earlier, but at the detriment to accuracy. All the other approaches perform somewhat similarly in a margin of 10 %–20 % accuracy rating.

Looking at the different plots in Fig. 10 also shows the influence of restricting outliers with different upper bounds: Using *any* of the two upper bound approaches to restrict outliers improves the accuracy compared to no outlier restriction at all for most approaches. But how strong this influence is depends on how stable the approach itself is against outliers. While linReg and avg, which are both heavily influenced by outliers, clearly improve their accuracy, med and min, which are mathematically robust to outliers, are only marginally affected. The fact that linRegQuant improves only marginally shows that using only the neighbors within the interquantile range for the estimation already restricts the outliers effectively.

There is only one of our approaches, min, that even has a slightly negative effect on accuracy when restricting outliers: with less outliers among the neighbors, minimum takes more older neighbors into account, which lowers its estimation leading to a slightly less accurate result.

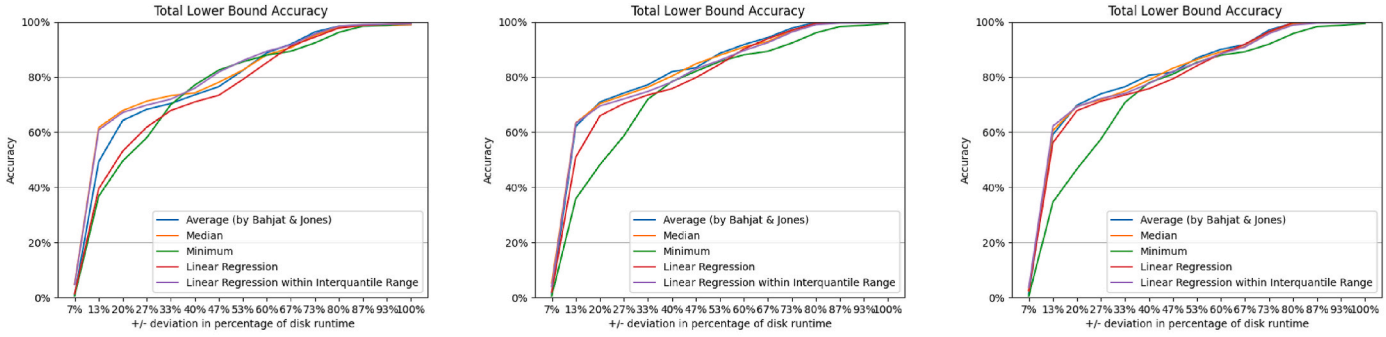
Comparing naïve to BJ yields another interesting insight: even though BJ outperforms naïve in terms of accuracy and precision, their suitability for restricting outliers is comparable in terms of accuracy. This seems counter-intuitive at first but can be easily explained: naïve has lower precision, so it sometimes excludes neighbors that were

[5] In fact, additional plots show that category 10 yields a significantly lower precision than the other categories for every method estimating a lower bound in our experiments. These additional plots are published together with the corpus (Uthoff et al., 2025).




(a) Lower bound without considering any upper bound (b) Lower bound considering BJ (c) Lower bound considering naïve

**Fig. 9.** Precisions of various methods for estimating the lower bound in relation to k neighbors.




(a) Lower bound without considering any upper bound (b) Lower bound considering BJ (c) Lower bound considering naïve

**Fig. 10.** Accuracies of various methods for 20 neighbors.

created before file fragment was deleted. This benefits methods that slightly overestimate the true creation date, e.g., linReg or avg, but slightly penalizes methods underestimating the date, e.g., min.

We now turn to the influence of the number of $k$ nearest neighbors on the lower bound accuracy. The results are shown in Fig. 11 for a deviation of ±20 % of the runtime of the disk image. We expected the following behavior: For low numbers, the accuracy should improve with an increasing number of neighbors because outliers have less of an influence than the "intended" neighbors. But for a high number of neighbors, there should be a point where the accuracy starts to decrease because the "intended" neighbors become older as more of them are added; thus, the estimated creation date starts to decrease.

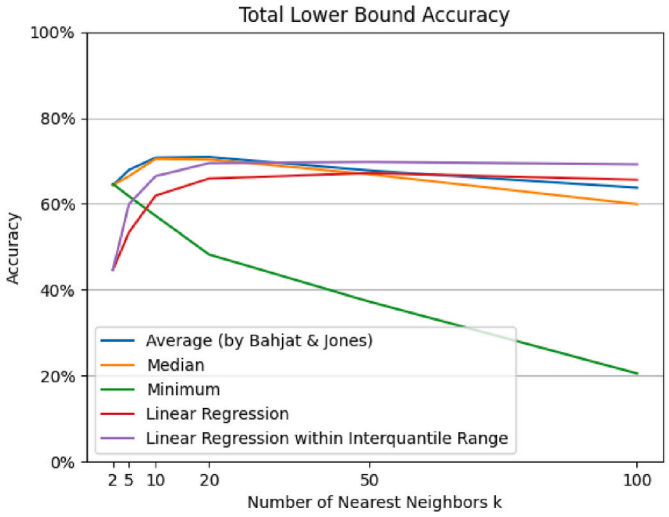



**Fig. 11.** Accuracies of various methods considering varying $k$ with a deviation of ±20 % of the complete runtime of the image using BJ.

In fact, both effects can be seen for the majority of the approaches: For avg, med, linReg and linRegQuant, the accuracy starts to increase at first and starts to decrease at individual points: the optimal $k$ varies between 10 and 50 for the these methods. Only min seems to decrease in accuracy directly with more than 2 neighbors. This is likely because the outliers are only impactful if no "intended" neighbor is part of the $k$ nearest neighbors. At the same time, neighbors becoming older has a tremendous effect. As such, the optimum is so early that any increase in the beginning remains unnoticed.

### 4.5. Summary of results

To summarize our results, we can conclude that BJ outperforms naïve in terms of accuracy as well as precision and can thus be regarded as the better option for lifetime estimation as well as for lifetime bounding.

For the lower bound approaches, we must differentiate: min has a very high precision and is thus the best candidate for lifetime bounding despite its low accuracy. But from our results it seems hard to achieve a really high accuracy and do lifetime estimation really well. While there are clearly worse ways to calculate the lower bound, there is no clearly better method than Bahjat and Jones.

## 5. Limitations

As presented in the previous section, the corpus has greatly facilitated the way in which file fragment dating in general and Bahjat and Jones (2019) in particular can be understood. Because it maps commands in the playbook to instructions for a concrete file system implementation, the results of the measurement should be authentic and incorporate special behavior like *file tunelling*.

By no means was the corpus constructed to represent typical, i.e., representative, file system activity. For this, information on how often which file system operation is executed in practice is missing. The dataset of real world disk images collected by Garfinkel et al. (2009) is large but lacks ground truth information about what really happened. In contrast, our corpus is rather homogeneous regarding file operations per category, a fact that clearly influences measurements. Having corpus categories that attempt to mix file operations and run longer would be more challenging for lifetime bounding methods, but finding good measures for representativeness is a research challenge in itself. Still, our corpus is useful for showing the effect of certain operations on the chronology of file fragments.

Despite the large number of images in the corpus, it still lacks several scenarios that represent typical edge cases for file fragment dating, e.g., images after applying *defragmentation* or images containing files that were copied from other disks. We limited our corpus to operations that are connected to file creation even though other operations (like file renaming) may influence timestamps as well.

An additional limitation of the corpus may be the relatively small number of files and file operations used to create the individual images. While this can be easily increased (by downloading the playbook files, extending them and rerunning the experiments with `fsstratify`), we believe that the effects of individual classes of file operations on file fragment dating can be readily observed. This is also evident in the differentiated discussion on the file fragment dating method from Bahjat and Jones (2019) in Section 4.

## 6. Conclusion

In this paper, we presented a slightly refined version of the `fsstratify` tool and used it to produce a corpus of NTFS file system images for testing and evaluating digital stratigraphic methods. The corpus covers homogeneous edge cases of file system operations and can therefore be used to benchmark and compare different methods because it comes with a ground truth. We demonstrated the usefulness of the corpus (and indirectly of the tool that created it) by evaluating the dating method of Bahjat and Jones (2019) and comparing it to slight variations of it.

Future work should expand the corpus and consider further edge cases like copying files across volumes or the application of defragmentation operations. Images with a representative mixture of many file operations would also be helpful, although a proper definition of representativeness remains absent.

Regarding `fsstratify`, we are implementing features allowing external files to be included in a simulation, which is relevant when `fsstratify` is used to generate data sets to evaluate file carving or other data recovery methods. Moreover, we want to include *complex operations* reflecting more high-level events. Such events include actions like “an editor saving a file” that comprise more than one of the operations described here (e.g., write the new content to a temporary file and move it to the original file). Operations like these ease the evaluation of effects like file tunneling, which Casey (2018) has already observed in his work, where he introduced the concept of digital stratigraphy.

Apart from complex operations, encryption is another interesting problem to tackle. Currently, fsstratify does not support encryption—neither at the volume level (e.g., via LUKS) nor file system inherent encryption, as provided by, for example, ZFS. However, incorporating encryption support at both levels is feasible without requiring changes to the core architecture of fsstratify, and this enhancement is planned for a future release. For the evaluations presented in this paper, encryption has only a limited impact. The method of Bahjat and Jones needs three types of information: knowledge of the existence of an interesting fragment in a certain block, the location of files, and the timestamps of neighboring files. With file-based encryption, this metadata is often accessible without the key; with volume-based encryption, the key is generally required. In controlled experiments as we conducted them, the key is known; in real-world investigations, the key is a prerequisite for most analyses—Bahjat and Jonesis no exception in this regard.

## Acknowledgments

We thank the anonymous reviewers for their comments and our shepherd Wietse Venema for his support in finalizing this paper. Work was supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) as part of the Research and Training Group 2475 “Cybercrime and Forensic Computing” (grant number 393541319/GRK2475/2-2024).

## References

Bahjat, A., Jones, J., 2023. File allocation chronology and its impact on digital forensics. In: 2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC). IEEE, pp. 612–618. https://doi.org/10.1109/CCWC57344.2023.10099265.

Bahjat, A.A., Jones, J., 2019. Deleted File Fragment Dating by Analysis of Allocated Neighbors, vol. 28, pp. S60–S67. https://doi.org/10.1016/j.diin.2019.01.015.

Bojic, N., Lambertz, M., Hilgert, J.N., 2020. Fsstratify: a Framework to Generate Used File Systems. Poster at DFRWS EU 2020.

Bouma, J., Jonker, H., van der Meer, V., Aker, E.V.D., 2023. Reconstructing timelines: from NTFS timestamps to file histories. In: Proceedings of the 18th International Conference on Availability, Reliability and Security, ARES 2023, Benevento, Italy, 29 August 2023- 1 September 2023. ACM, pp. 154:1–154:9. https://doi.org/10.1145/3600160.3605027.

Casey, E., 2018. Digital stratigraphy: contextual analysis of file system traces in forensic science. J. Forensic Sci. 63, 1383–1391.

Chow, K., Law, F., Kwan, M., Lai, P., 2007. The Rules of Time on Ntfs File System, pp. 71–85. https://doi.org/10.1109/SADFE.2007.22.

Dreier, L.M., 2025. Evaluation scripts for fragment dating. https://github.com/Lisa-0x3/file-fragment-dating-evaluation.

Garfinkel, S.L., Farrell Jr., P.F., Roussev, V., Dinolt, G.W., 2009. Bringing science to digital forensics with standardized forensic Corpora. Digit. Invest. 6, S2–S11. https://doi.org/10.1016/J.DIIN.2009.06.016.

Garfinkel, S.L., Parker-Wood, A., Huynh, D., Migletz, J.J., 2010. An automated solution to the multiuser carved data ascription problem. IEEE Trans. Inf. Forensics Secur. (5), 868–882. https://doi.org/10.1109/TIFS.2010.2060484.

Harris, E.C., 1989. In: Priciples of Archaeological Stratigraphy, second ed. Academic Press.

Horsman, G., 2019. Tool testing and reliability issues in the field of digital forensics. Digit. Invest. 28, 163–175. https://doi.org/10.1016/J.DIIN.2019.01.009.

Karresand, M., 2023. Digital Forensic Usage of the Inherent Structures in NTFS. Norwegian University of Science and Technology, Trondheim, Norway. Ph.D. thesis. https://hdl.handle.net/11250/3069265.

Lambertz, M., Rybalka, M., 2025. Fsstratify. https://github.com/fkie-cad/fsstratify.

Metz, J., 2021. Pearls and pitfalls of timeline analysis. URL: https://osdfir.blogspot.com/2021/10/pearls-and-pitfalls-of-timeline-analysis.html.

Nemetz, S., Schmitt, S., Freiling, F.C., 2018. A standardized corpus for SQLite database forensics. Digit. Invest. 24 (Suppl. ment), S121–S130. https://doi.org/10.1016/J.DIIN.2018.01.015.

Palmbach, D., Breitinger, F., 2020. Artifacts for detecting timestamp manipulation in NTFS on windows and their reliability. Digit. Invest. 32 (Suppl. ment), 300920. https://doi.org/10.1016/J.FSIDI.2020.300920.

Schneider, J., Eichhorn, M., Dreier, L.M., Hargreaves, C., 2024. Applying digital stratigraphy to the problem of recycled storage media. Forensic Sci. Int.: Digit. Invest. 49, 301761. https://doi.org/10.1016/j.fsidi.2024.301761. DFRWS USA 2024 - Selected Papers from the 24th Annual Digital Forensics Research Conference USA.

Uthoff, J., Lambertz, M., Rybalka, M., Dreier, L.M., 2025. The digital forensic corpus for temporal stratigraphic file system analysis. https://zenodo.org/records/16637419.

Vanini, C., Hargreaves, C.J., van Beek, H., Breitinger, F., 2024. Was the clock correct? Exploring timestamp interpretation through time anchors for digital forensic event reconstruction. Forensic Sci. Int.: Digit. Invest. 49, 301759. https://doi.org/10.1016/j.fsidi.2024.301759. DFRWS USA 2024 - Selected Papers from the 24th Annual Digital Forensics Research Conference USA.

Willassen, S.Y., 2008. Finding evidence of antedating in digital investigations. In: Proceedings of the the Third International Conference on Availability, Reliability and Security, ARES 2008, March 4-7, 2008, Technical University of Catalonia. IEEE Computer Society, Barcelona , Spain, pp. 26–32. https://doi.org/10.1109/ARES.2008.149.