Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

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

DFRWS USA 2025 - Selected Papers from the 25th Annual Digital Forensics Research Conference USA

An extensible and scalable system for hash lookup and approximate similarity search with similarity digest algorithms

Daniel Huici [a], Ricardo J. Rodríguez [b,*], Eduardo Mena [a]

[a] Dpto. de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, Spain
[b] Aragón Institute of Engineering Research (I3A), Universidad de Zaragoza, Spain

ARTICLE INFO

Keywords:
Approximate matching
Similarity hashing
Similarity digest algorithms
Hash lookup
Similarity search

ABSTRACT

Efficient management and analysis of large volumes of digital data has emerged as a major challenge in the field of digital forensics. To quickly identify and analyze relevant artifacts within large datasets, we introduce APOTHEOSIS, an approximate similarity search system designed for scalability and efficiency. Our system integrates approximate search techniques (which allow searching for a match on a close value) with Similarity Digest Algorithms (SDA; which capture common features between similar elements), using a space-saving radix tree and a graph-based hierarchical navigable small world structure to perform fast approximate nearest neighbor searches. We demonstrate the effectiveness and versatility of our system through two key case studies: first, in plagiarism detection, demonstrating the effectiveness of our system in identifying similar or duplicate documents within a large source code dataset; then, in memory artifact detection, showing its scalability and performance in processing large-scale forensic data collected from various versions of Microsoft Windows. Our comprehensive evaluation shows that APOTHEOSIS not only efficiently handles large datasets, but also provides a way to evaluate the performance of various SDA and their approximate similarity search in different forensic scenarios.

1. Introduction

The exponential growth of digital data and the proliferation of various digital devices (such as computers, smartphones, tablets, and Internet of Things devices, to name a few) have significantly increased the complexity of digital forensic investigations. Analysts often must examine large amounts of data in which every file, application, system log, and network packet could contain valuable evidence. Every single digital evidence (also called *digital artifacts*) may shed light on the nature, scope, and impact of a security incident (Johansen, 2022). Identifying relevant information in this vast amount of data is, however, a time- and resource-intensive process.

The convergence of two distinct –yet deeply interconnected– technologies, such as approximate search techniques and approximate similarity matching algorithms, can help in this process. Approximate search techniques identify very close matches and offer the potential to speed up the identification within large data sets, relaxing the constraint of *finding an exact match*. Similarly, approximate similarity matching algorithms provide the means to discern common features shared between similar artifacts. In this paper, we focus on approximate similarity matching algorithms that treat input (the artifacts) as a byte stream, process them without any interpretation of the data, and utilize an intermediate representation (i.e., a digest or fingerprint) to *summarize* the input. We define this kind of algorithms as *similarity digest algorithms* (SDA).[1] SDA allows us to find items that are not necessarily the same, but simply similar in some way (Breitinger et al., 2014a).

However, the effectiveness of different SDA can vary significantly depending on the type of forensic data (Martín-Pérez, 2022). For instance, an SDA that performs well in detecting similarities in textual documents may not be as effective in analyzing binary executables or memory dumps. Therefore, there is a need for systems that can adapt their similarity search mechanisms based on the nature of the objects analyzed, ensuring effective detection and analysis across different types of digital artifacts.

In this paper, we present how APOTHEOSIS, an efficient approximate search system, can address the problem of searching similarity digests across a wide range of digital artifacts (Huici et al., 2025). To achieve this goal, APOTHEOSIS leverages two data structures: a tree

* Corresponding author.
E-mail address: rjrodriguez@unizar.es (R.J. Rodríguez).

[1] In this paper, we use the SDA interchangeably as a singular and plural acronym.

https://doi.org/10.1016/j.fsidi.2025.301930

data structure (in particular, a radix tree (Morrison, 1968)), which is employed for efficient hash lookup search; and a custom implementation of Hierarchical Navigable Small World (HNSW) (Malkov and Yashunin, 2020), which is a graph-based approximate search method designed to provide fast, approximate nearest neighbor search in high-dimensional spaces. By integrating these frameworks with support for multiple SDA, our system enables fast and efficient approximate nearest neighbor searches on similarity digests, facilitating the rapid identification of related artifacts within large datasets. Furthermore, its extensible design enables forensic analysts to evaluate and compare the performance of different SDA and approximate search in various contexts, helping to select the most appropriate algorithms for specific forensic tasks.

Our system is fully extensible and adaptable to a wide range of digital forensics problems, primarily acting as a tool for exact hash lookups and nearest neighbor searches. To demonstrate its versatility in different forensic contexts, we perform a comprehensive evaluation through two case studies:(i) we apply APOTHEOSIS with SDA such as ssdeep and TLSH to detect duplicate or plagiarized documents within a large source code dataset. We also compare APOTHEOSIS with other approximate similarity search techniques, specifically MinHash (Broder, 1997) with Locality-Sensitive Hashing (LSH) (Leskovec et al., 2020). We evaluate the search performance through standard metrics such as accuracy, recall, precision, and F1-score, demonstrating how our system can help evaluate the effectiveness of different SDA and approximate similarity searches in analyzing text-based artifacts; (ii) we adapt our system to handle a large dataset of memory artifacts collected from various 64-bit versions of Microsoft Windows, from Windows 7 onward. This case study allows us to evaluate the performance and scalability of different SDA in the context of binary data analysis, further showing how APOTHEOSIS can efficiently handle large datasets.

Our experiments demonstrate that APOTHEOSIS not only efficiently identifies similar artifacts, but also provides valuable insights into the performance of different SDA on various forensic tasks. We selected ssdeep and TLSH for the case studies to illustrate the versatility of our system in different forensic contexts, but it can easily incorporate other SDA depending on the specific needs of the analyst.

In summary, the contribution of this paper is three-fold:

(i) We present APOTHEOSIS, a novel approximate similarity search system that uniquely combines a space-saving radix tree with a custom Hierarchical Navigable Small World (HNSW) graph implementation. This integration enables fast and scalable approximate nearest neighbor searches over similarity hashes, effectively addressing the challenge of efficiently managing and analyzing large-scale digital forensic datasets. We have released the source code of our system under the GNU/GPLv3 license (Huici et al., 2025) to encourage collaboration, further development, and integration into existing forensic workflows, advancing research and practice in the field.

(ii) Through our case studies, we illustrate the adaptability of APOTHEOSIS to various forensic problems, including text-based artifact analysis and binary data analysis, highlighting its flexibility in handling different types of digital artifacts. We demonstrate how our system can be used to evaluate and compare the performance of approximate similarity searches on different SDA, thus providing insights that help select the most effective algorithms for specific tasks.

(iii) We conduct a comprehensive evaluation of APOTHEOSIS, assessing its performance and scalability using a large dataset of approximate hashes comprising millions of records. Our results demonstrate that our system significantly improves the efficiency of similarity searches, maintaining low computational overhead.

The rest of this paper is as follows. Section 2 delves into the foundational concepts that underpin our work. Section 3 introduces the architecture of APOTHEOSIS and provides insights on its design and functionality. Section 4 presents the results of the comprehensive evaluation of our system. In Section 5, we critically examine the threat model and the limitations of our approach. Section 6 discusses related work, and finally Section 7 concludes the paper and sets out future research.

2. Background

In this section, we present the basic concepts needed to understand our work. We begin by discussing similarity digest algorithms, which are essential to compare digital artifacts based on their content. We then describe efficient search methods used (in particular, radix trees and HNSW) that enable fast searching and matching within large data sets.

2.1. Similarity digest algorithms

Similarity Digest Algorithms (SDA) are a class of *approximate matching algorithms* (or *approximate similarity matching algorithms*) (Breitinger et al., 2014a) used to identify similarities between digital artifacts, such as files or memory dumps, even when they are not identical. SDA work by processing input data (treated as byte streams) without interpreting the underlying content. They generate an intermediate representation, often called a *fingerprint*, *hash*, or *digest*, that summarizes the essential features of the input.

Unlike traditional cryptographic hash functions (Katz and Lindell, 2015) that produce completely different results even for minor changes in the input data (known as the *avalanche effect* (Webster and Tavares, 1986)), SDA generate digests that reflect the similarity between inputs. This property allows forensic analysts to detect related or derived artifacts, which is essential in scenarios such as malware detection, plagiarism detection, and identification of modified files, to name a few. Comparison between the digests produces a similarity score, which indicates the degree of similarity between the original artifacts.

According to Martín-Pérez et al. (2021), SDA can be classified based on their hash generation methods as: (i) *Feature Sequence Hashing* (FSH), which encompasses algorithms that split the input into features and map them to hash values. Similarity is assessed by comparing the feature sequences between different digests; An example of an FSH algorithm is ssdeep, which calculates context-triggered piecewise hashes and is effective at detecting embedded or concatenated files (Kornblum, 2006); (ii) *Byte Sequence Existence* (BSE), which comprises algorithms that focus on detecting the presence or similarity of specific byte sequences (often referred to as *blocks*) within the input data. The similarity score is then determined by comparing the number of shared sequences or blocks between digests; and (iii) *Locality-Sensitive Hashing* (LSH), which assigns similar input objects to the same bins (or buckets) with high probability. TLSH (*Trend Micro Locality-Sensitive Hash)* is an example of an LSH algorithm that is commonly used in malware detection.

In this particular work, we combine APOTHEOSIS with two well-known SDAs of different types, specifically: (i) ssdeep, an FSH algorithm proposed by Kornblum (2006), initially designed for spam detection but widely used in digital forensics. ssdeep generates hashes of variable length, up to 148 characters, and produces a similarity score between 0 and 100, where 0 indicates no similarity and 100 indicates identical inputs; (ii) TLSH, an LSH algorithm introduced by Oliver et al. (2013) and primarily used for malware detection. TLSH generates fixed-length hashes of 70 characters (excluding version flags) and calculates a similarity score based on the distance between digests. A score of 0 means that the entries are nearly identical; while higher scores indicate greater differences.

2.2. Efficient search methods

As volume of digital data grows exponentially, efficiently searching and identifying similar items within massive data sets becomes increasingly difficult. To address this, our system combines two data structures that enable fast and scalable searches: the radix tree and the HNSW.

2.2.1. Radix tree

A radix tree, also known as *radix trie*, *compressed trie*, or *compact prefix tree*, is a space-saving data structure designed to store and search strings. It optimizes storage by merging nodes with common prefixes, effectively reducing redundancy and saving memory. Radix trees are particularly useful in applications that require fast retrieval of key–value pairs, such as dictionaries, IP routing tables (Morrison, 1968), and file systems, to name a few.

In a radix tree, each node represents a sequence of characters (a substring) of the keys being stored. Edges correspond to the substrings that differentiate the keys. This structure allows for efficient search operations by traversing the tree based on the characters of the query key, allowing one to quickly locate the desired key or determine its absence. When inserting a new key–value pair, the tree dynamically adjusts its structure, merging nodes with common prefixes (if necessary) to maintain its compact form. Deletion involves removing nodes as necessary, ensuring that the tree remains efficient and well-organized.

The computational complexity of search and insert operations in a radix tree is $\mathscr{O}(k)$, where k is the key length (Morrison, 1968). This means that the time to search for a key depends on the length of the key, not the number of keys stored, making radix trees highly efficient for exact matching in scenarios with a large number of strings.

2.2.2. Hierarchical navigable small world (HNSW)

A commonly employed method for searching for information is K-Nearest Neighbor Search (K-NNS), which operates under the assumption that a defined distance function exists between data elements. K-NNS aims to identify the K elements within a dataset that minimize the distance to a specified query. A straightforward implementation of K-NNS entails computing distances between the query and every data element within the dataset, and then selecting those elements with the shortest distances. Unfortunately, the computational complexity of this naïve approach increases linearly with the number of elements, rendering it impractical for large-scale datasets (e.g., its time complexity in the worst-case is $\mathscr{O}(N)$, where N is the number of elements in the dataset). Consequently, there has been significant interest in developing rapid and scalable K-NNS algorithms.

Additionally, a number of challenges and problems arise when working with high-dimensional data. To overcome *the curse of dimensionality* in exact solutions for K-NNS (Yianilos, 1993), the concept of Approximate K-Nearest Neighbors Search (K-ANNS) was proposed, which seeks to find a set of K data points that are close to the query point but not necessarily the absolute closest. This relaxation can be useful in situations where finding the exact nearest neighbors is computationally expensive or impractical, such as in high-dimensional data spaces. There are several types of K-ANNS methods, such as probability-based (e.g., LSH (Leskovec et al., 2020)), tree-based (e.g., KD-Tree (Bentley, 1975)), or graph-based (e.g., NN-Descent (Dong et al., 2011)).

In this work, we use Hierarchical Navigable Small World (HNSW) (Malkov and Yashunin, 2020), a widely adopted graph-based K-ANNS method. HNSW is based on the Navigable Small World (NSW) (Watts and Strogatz, 1998), which constructs a proximity graph with short and long links, enabling rapid navigation through local and distant regions of the data space.

HNSW introduces a hierarchical structure by organizing data points into multiple layers. The *base layer* (or *ground layer*) contains all the elements (e.g., *Layer 0* in Fig. 1), while higher layers house increasingly smaller subsets. This multi-layered graph balances efficiency and accuracy, allowing searches to start from a sparse top layer and refine results as they descend to the denser base layer.

To gain a broader perspective on how our approach compares to existing methods, a detailed discussion of related work and alternative approaches is provided in Section 6. Below we explain in a simple and brief way how insertion and search work in HNSW.

Construction/insertion. When adding a new data point, the algorithm determines the maximum level for the node based on a probabilistic model. The new node connects to its K nearest neighbors in each layer up to its maximum level, establishing both short- and long-range connections. This process maintains the small-world property of the graph, ensuring efficient navigability. To enforce this property, it may prune weaker connections and adjust levels in the graph structure. The construction complexity scales as $\mathscr{O}(N\log N)$, while the insert complexity scales as $\mathscr{O}(\log N)$, where N is the number of nodes (Malkov and Yashunin, 2020).

Search. To find approximate nearest neighbors for a query point, the search starts at the highest layer with the entry point node. The algorithm traverses the graph exploring the neighboring nodes that are closest to the query at that layer, moving progressively to lower layers. At each step, it maintains a candidate list of possible nearest neighbors, pruning nodes that are farther away. This hierarchical traversal allows the algorithm to efficiently reduce the search space and find approximate nearest neighbors with a time complexity of $\mathscr{O}(\log N)$ (Malkov and Yashunin, 2020).

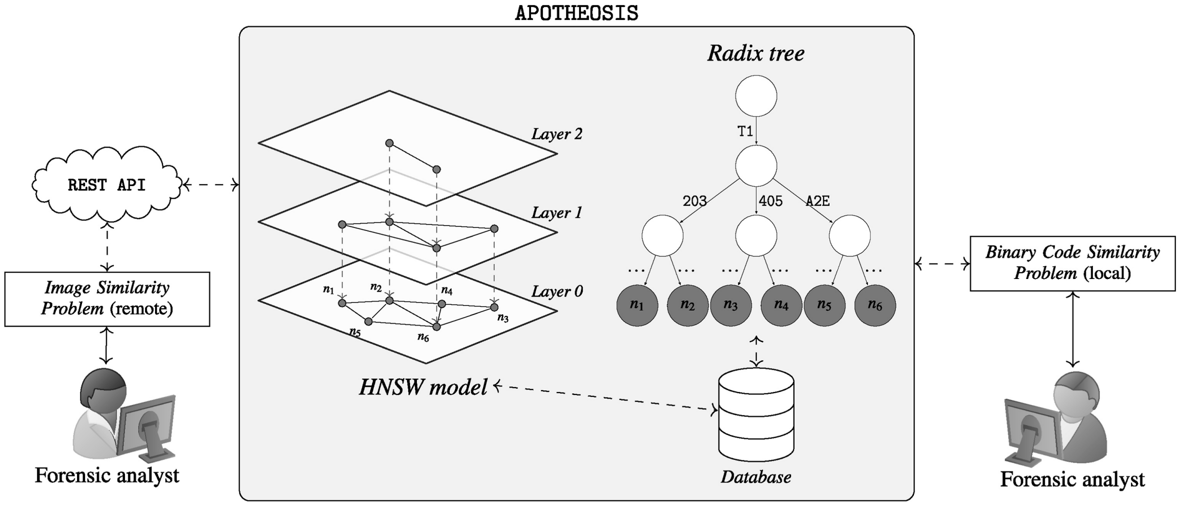


Fig. 1. High-level overview of different use cases of APOTHEOSIS in digital forensics.

3. APOTHEOSIS: system architecture

Efficient similarity detection in forensics requires a system capable of handling both exact and approximate searches across diverse datasets. APOTHEOSIS (APprOximaTe searcH systEm Of Similarity dIgeSts) achieves this by integrating a radix tree for fast exact hash lookups and a HNSW graph for scalable approximate nearest neighbor search. While these data structures are well established, their standard implementations are primarily designed for continuous vector spaces, making them unsuitable for hash-based data and the similarity scores generated by SDAs. In this section, we describe how APOTHEOSIS adapts these structures to forensic similarity detection, enabling efficient retrieval of similar files across a variety of digital artifacts.

Fig. 1 shows a high-level overview of APOTHEOSIS and its potential use cases in digital forensics. The system is designed to be flexible and adaptable, allowing it to be used in various scenarios. By offering local deployment as a software library and remote access via a REST API, our system provides flexibility in how it can be integrated and used within different forensic analysis workflows.

As described earlier, APOTHEOSIS leverages two data structures that work together: a custom implementation of radix tree and HNSW. These data structures are combined with similarity digests to enable efficient approximate similarity searches. The APOTHEOSIS database stores all hashes and associated extended information for each hash. By keeping this information separate from the HNSW structure, we maintain a low space complexity for the HNSW graph.

The radix tree is used to store similarity digests and efficiently retrieve exact matches. Unlike hash tables, it allows for prefix-based searches, making it well suited for detecting partial or hierarchical similarities within structured forensic data. In parallel, HNSW is used for approximate similarity searches, offering a trade-off between speed and accuracy on large-scale datasets. However, existing HNSW implementations assume continuous vector spaces and rely on distance metrics such as Euclidean or cosine similarity. Since SDAs generate discrete hash values with algorithm-specific scoring functions, APOTHEOSIS adapts HNSW to use SDA-derived similarity scores instead of traditional distance-based metrics.

The system works in two phases: indexing and querying. During indexing, similarity digests are extracted using a modular SDA interface and stored in the radix tree for exact searches. At the same time, digests are inserted into the HNSW graph, where edges to similar nodes are created based on the calculated similarity scores. The query phase first checks the radix tree for exact matches. If no exact match is found, an approximate similarity search is performed using HNSW, retrieving the K nearest neighbors based on the SDA similarity scores. This combined approach ensures high-speed retrieval of similar files, improving forensic efficiency in identifying duplicate or related artifacts in diverse datasets, from textual documents to binary executables.

APOTHEOSIS supports two types of search methods for added flexibility: (i) *Nearest Neighbors-Based Search*: To identify the K most similar nodes to a given query node (where $K \in \mathbb{N}_{>0}$), the system performs an approximate nearest neighbor search starting at the highest layer of the HNSW graph from the entry point node. At each level, it evaluates neighboring nodes to find approximate closest matches based on the similarity score, refining the set of candidate nodes as it moves down through the layers. A priority queue is used to manage candidates efficiently, prioritizing nodes that are closer to the query. An improvement in our system, compared to the original HNSW algorithm (Malkov and Yashunin, 2020), is that it also considers the neighbors of the candidate nodes during the traversal; this increases the probability of finding nodes that are closer to the query, thus improving the quality of the approximation. Upon reaching the ground layer (level 0), the algorithm selects the final K approximate nearest neighbors; and (ii) *Threshold-Based Search.* In threshold-based searches, the system retrieves nodes whose similarity scores meet a specific condition relative to a threshold value $t > 0$. The search parameters include the query node (the hash for which similar nodes are searched), the threshold value t (a similarity score threshold), and the comparison mode (which specifies whether to retrieve nodes with similarity scores above or below the threshold t). The search process is similar to nearest neighbor search, but with key differences: at the base layer, only nodes that meet the threshold condition are added to the candidate queue. As before, it also analyzes the neighbors of the candidate nodes and selects those that are closest to the query and meet the given threshold criteria.

To optimize performance, a sensitivity analysis is recommended for key HNSW parameters since their settings play a crucial role in balancing accuracy, memory consumption, and computational overhead. The parameters M and $M_{\max}$, which control the graph connectivity, affect recall by increasing the number of connections per node, thereby improving search accuracy at the expense of higher memory usage and longer search times. The parameter ef, which regulates the number of candidate nodes explored during queries and affects indexing efficiency, improves recall but results in longer query times. Furthermore, $M_{\max0}$, which defines connectivity at the lowest layer, also influence the overall system performance. Since these parameters directly affect search quality and resource constraints, a systematic sensitivity analysis is necessary to achieve an optimal balance tailored to the requirements of the forensic dataset.

Note that the combination of the radix tree and HNSW facilitates efficient processing of large data sets, while the system's ability to support multiple SDAs ensures adaptability across diverse forensic applications. By integrating exact and approximate search mechanisms within a modular framework, APOTHEOSIS offers a flexible and scalable solution for forensic similarity detection. The only requirement is that an intermediate representation (such as a similarity digests or hash) can be generated for the data items, allowing the system to work across a variety of digital artifacts.

APOTHEOSIS is open-source software released under the GNU/GPLv3 license (Huici et al., 2025). The availability of the source code promotes transparency and encourages collaboration within the research community. The system's REST API interface (Richardson and Ruby, 2007) simplifies deployment and integration into existing forensic analysis workflows, making it accessible to a broader audience.

4. Evaluation

To evaluate the effectiveness and applicability of APOTHEOSIS in the field of digital forensics, we designed a comprehensive evaluation considering document duplication detection and performance on large digital forensic datasets. Our goal is to demonstrate how our system can enhance forensic investigations by improving the speed of similarity searches on different types of digital artifacts.

We focus on the following research questions to guide our evaluation:

RQ1.–*How effective is APOTHEOSIS in detecting duplicate content or modified versions of original documents?* (see *§* 4.1).
RQ2.– *How does APOTHEOSIS perform in terms of performance and scalability when handling large digital forensic datasets?* (see *§* 4.2).

To answer these questions, we first present the datasets constructed for the experiments, as well as the experimental setup and metrics used. We then answer each question.

Datasets, Metrics, and Experimental Setup. To address RQ1, we used the dataset provided in (Ljubovic and Pajic, 2020), which contains student assignment submissions from two introductory programming courses at the University of Sarajevo (Bosnia and Herzegovina). This dataset includes a total of 43,792 source code files, making it well suited for evaluating the effectiveness of APOTHEOSIS in detecting document duplication or forgery in forensic contexts.

Since the provided ground truths do not specify exact similarity scores or thresholds used to classify plagiarism, we generated a new ground-truth dataset. We computed similarity scores for each pair of files using ssdeep and TLSH, following a cleaning process to exclude files too small to generate valid similarity digests.

For each SDA, we built an APOTHEOSIS model using the new ground-truth dataset and configured with $M = ef = M_{max} = M_{max0} = 16$. Since APOTHEOSIS primarily operates as a hash lookup mechanism and an approximate nearest neighbor search tool for hash-based comparisons, its effectiveness is closely tied to the performance of the underlying SDA.

To provide a comprehensive evaluation of our tool, we perform a comparative analysis with MinHash, a simply and widely recognized technique for estimating the Jaccard index (Broder, 1997; Leskovec et al., 2020), which quantifies the overlap between generated signatures. When combined with LSH, MinHash facilitates efficient K-ANNS by grouping similar items into the same hash buckets, significantly reducing computational complexity for large-scale similarity detection tasks. We refer to this combination as MinHashLSH. Following the same experimental procedures used with SDA, we generate MinHash signatures for each file using $N = 72$ permutations and compute the Jaccard similarities between all file pairs to establish the ground truth.

We evaluate the performance of these models using standard metrics: accuracy ($Acc = \frac{TP+TN}{TP+TN+FP+FN}$), precision ($Prec = \frac{TP}{TP+FP}$), recall ($Rec = \frac{TP}{TP+FN}$), and F1-score ($F1 = 2 \cdot \frac{Prec \cdot Rec}{Prec+Rec}$). Here, *TP* represents the true positives (correctly detected plagiarism cases), *FP* denotes the false positives (incorrect plagiarism detections), *TN* refers to the true negatives (correctly identified non-plagiarized files), and *FN* corresponds to the false negatives (undetected plagiarism cases).

For RQ2, we used a dataset containing Windows system modules (shared libraries, kernel drivers, and system executables) collected from different versions of Microsoft Windows. Specifically, we considered Windows 7, Windows 8.1, Windows Server 2012 R2, Windows Server 2016; Windows Server 2019 on 64 bit architectures. For each operating system, we created a virtual machine and perform a fresh installation of Windows to avoid the possible presence of malware or unwanted software. We then ran a program that iterated through the Windows system folder, loaded each system dynamic shared library into its process address space, and dumped the memory regions associated with the library that contained binary code to disk at a page granularity, along with its associated metadata. To accomplish this, we relied on the Windows Memory Extractor tool (Fernández-Álvarez and Rodríguez, 2022).

In Windows terminology, a module refers to the representation of the process memory space of an executable file or a shared library (Microsoft, 2021), while a *page* is a fixed-length block (typically 4096 bytes) into which the virtual memory of a Windows process is divided (Microsoft Docs, 2018).

We stored the relationship of each module to its pages and computed similarity hashes using ssdeep and TLSH via the SUM software library (Martín-Pérez et al., 2021). Additionally, where possible, we applied the best unrelocation method to normalize the raw bytes (Martín-Pérez et al., 2021). As metadata, we retrieved the file version, original and internal filenames, product and company names, legal copyright, byte size, and base address. All of this information is stored in a SQL database (MySQL). In total, we collected approximately 4.2 million pages belonging to about 37,500 Windows system files.

We chose this dataset to evaluate the system's performance and scalability in handling large forensic datasets because it closely reflects real-world scenarios. For instance, this dataset can be used to create a list of allowed SDA hashes, allowing the system to function as a detection mechanism that quickly identifies known, non-malicious artifacts, thereby improving the efficiency of forensic investigations (Pryde et al., 2018). Unlike the source code dataset used in RQ1, which only contain file names and similarity scores, here we retain more data for each node (such as module metadata, the operating system it originates from, and other binary file details). This dataset is more representative of actual systems used during forensic investigation, where multiple pieces of data can be associated with hashes. As metrics, we measured the average time taken to insert a new node, perform hash lookups, and retrieve K-NN of a given hash under different HNSW configurations. For this experiment, we consider only TLSH.

Let us recall that the dataset used for RQ2 serves only to illustrate the functionality of APOTHEOSIS, which is dataset independent and applicable to various forensic investigations.

All experiments were conducted on a personal computer with an Intel(R) Core(TM) i7-10700 processor @2.90 GHz. The system is equipped with 64 GB of DDR4 RAM clocked at 3200 MHz and runs the GNU/Linux Debian 11.7 operating system.

4.1. RQ1: effectiveness in document duplication detection

4.1.1. Performance metrics across varying thresholds

Fig. 2 presents the performance metrics for the APOTHEOSIS models with ssdeep and TLSH, as well as MinHashLSH, at different threshold values τ. Each threshold τ corresponds to a distinct ground-truth dataset, constructed by applying the corresponding threshold to the similarity scores on the new ground-truth dataset. In this way, each dataset contains only those file pairs that meet the specified threshold criteria, allowing us to evaluate how APOTHEOSIS performance metrics vary as the definition of "similar" or "duplicate" files becomes more or less strict. For the APOTHEOSIS models, we restrict the K-NN search operation to a maximum of $K = 8$ neighbors.

With the APOTHEOSIS models, the precision remains consistently high at 1.00 for both SDA across all τ, indicating that it does not generate false positives, in contrast to MinHashLSH.

Recall, on the other hand, shows gradual improvement as the threshold increases, meaning that more true positives are detected with stricter similarity requirements. This results in a similar upward trend for the F1-score. Accuracy also increases with higher thresholds, reflecting better overall performance in correctly identifying plagiarized and non-plagiarized files as more stringent ground-truth datasets are used.

MinHashLSH shows an overall improvement in performance metrics as similarity thresholds increase, but exhibits significant fluctuations, especially a noticeable drop in recall at $\tau = 80$. This non-linear behavior can be attributed to the probabilistic nature of these algorithms, where random permutations may not always optimally capture the true similarity between documents. Furthermore, we observe a key limitation in detecting exact matches (i.e., $\tau = 100$). This limitation arises from how

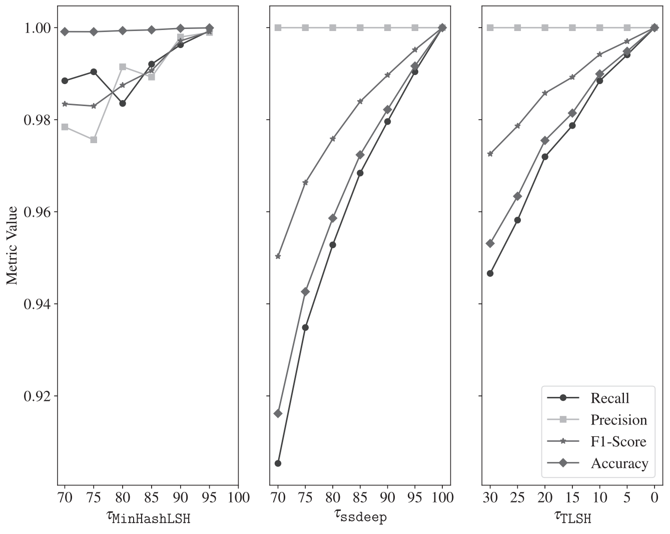


Fig. 2. Performance metrics comparison of APOTHEOSIS using ssdeep and TLSH, as well as MinHashLSH, at different thresholds τ.

LSH splits MinHash signatures into bands, relying on partial overlaps to identify candidates. At $\tau = 100$, all bands must match exactly, leaving no tolerance for discrepancies in the signatures. As a result, the probabilistic nature of LSH makes it unsuitable for detecting exact matches.

These findings indicate that as thresholds increase, all the techniques become more effective at detecting duplications. Specifically, with APOTHEOSIS, while both SDA improve with higher thresholds, TLSH consistently achieves higher recall and F1 scores in all τ evaluated, making it a more effective choice to detect duplications in text-based documents.

4.1.2. Number of plagiarized cases in MinHash*,* ssdeep*, and* TLSH *datasets*

Fig. 3 presents the cumulative distribution function of the detected matches per file for the MinHash, ssdeep, and TLSH datasets, considering $\tau = 70$. These datasets are the most inclusive in this experiment, as they also include all file pairs considered similar by the respective algorithms under stricter thresholds. Note that the number of files in each dataset, denoted by *N*, differs slightly. In the previous experiment, we imposed a maximum limit of $K = 8$ for the K-NN search operation in the APOTHEOSIS models. As shown in the figure, this limitation resulted in missed plagiarism cases, as the system does not fully capture additional cases beyond this limit. Specifically, there are cases where files have more than eight plagiarized matches, which are not fully represented in the results.

4.2. RQ2: performance and scalability on large datasets

We focus our evaluation on two key HNSW configuration parameters, *M* and *ef*, using $N = 10000$ and TLSH as SDA. The parameters M_{max} and M_{max0} were varied while holding the other settings constant to evaluate their impact on insert, K-NN lookup, and approximate K-NN (AKNN) search operations. Due to space limitations, we present a selection of representative plots that capture key trends observed across all experiments (the full set of results are available upon request).

4.2.1. Varying M with ef = 16

Increasing *M* from 4 to 32 shows a steady increase in insertion time across all values of M_{max} and M_{max0}, where M_{max0} exerts a more significant influence on performance. Similar trends are observed for both K-NN lookup and AKNN search operations. When $M = 4$, the insert operation time remains low, even at high values of M_{max} and M_{max0}. The AKNN search times level off as *M* increases, indicating diminishing returns at higher values. The K-NN search operation exhibits significantly lower runtime compared to the other operations, which show similar performance. This difference is primarily attributed to the traversal of the radix tree, where the query hash is found directly, eliminating the need for additional approximate search operations. Fig. 4a, b, and 4c shows the insert, lookup, and AKNN operation results for $M = ef = 16$, illustrating the general trend across all values of M_{max} and M_{max0}.

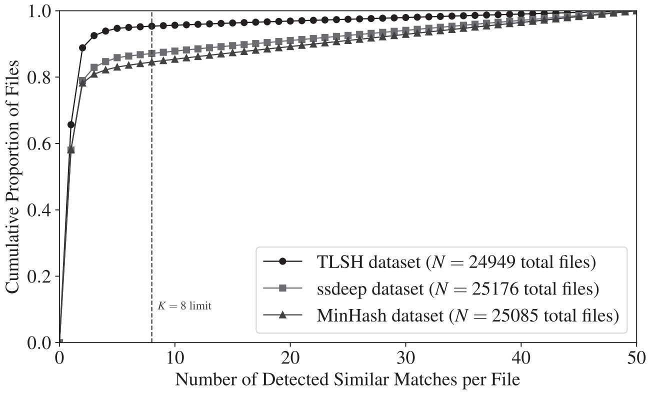


Fig. 3. Cumulative distribution of similar matches per file for ssdeep and TLSH datasets considering $\tau_{\mathrm{MinHashLSH}} = \tau_{\mathrm{ssdeep}} = 70$ and $\tau_{\mathrm{TLSH}} = 30$.

4.2.2. Varying ef with M = 16

As expected, increasing *ef* results in longer times for the insert, K-NN lookup, and approximate K-NN search operations. Higher *ef* values improve the accuracy of the approximate search by exploring more candidate nodes, but they also increase the operation times due to the additional computation. In contrast, low *ef* values reduce search time but at the cost of accuracy. Fig. 5 shows the approximate K-NN search operation results for $M = 16$ and $ef = \{2, 6, 14\}$.

4.2.3. Impact of N on time complexity

We perform a sensitivity analysis and employ a curve-fitting approach to validate the time complexities explained in Section 2.2.2. In this analysis, we considered three different system configurations CFG_i: (ef, M, M_{max}, M_{max0}) (specifically, CFG_1: (4, 4, 16, 16), CFG_2: (8, 8, 16, 16), and CFG_3: (8, 16, 16, 32)) with varying HNSW parameters while varying *N* from 100 to 50, 000, in increments of 100.

Fig. 6 shows, from left to right, the curve fitting plots for insert, hash lookup, and approximate K-NN search operations. All observed trends align with theoretical expectations. The insert and approximate K-NN search operations exhibit similar behaviors (in particular, a natural logarithmic growth trend as the dataset size increases), as they both involve traversing and updating the HNSW graph–although the approximate K-NN search requires some additional logic to complete. Among the configurations, CFG_3 consistently outperforms the others, which can be attributed to its higher *M* and M_{max} values. More connections lead to a more connected structure and faster convergence during these operations. The hash lookup operation, on the other hand, shows a nearly linear growth with a very low slope, indicating that it scales efficiently even with large datasets. In this case, CFG_3 was the worst-performing configuration. This is because it has more connections per node, which requires traversing and accessing more nodes during the lookup search, resulting in longer times.

5. Threat model and limitations

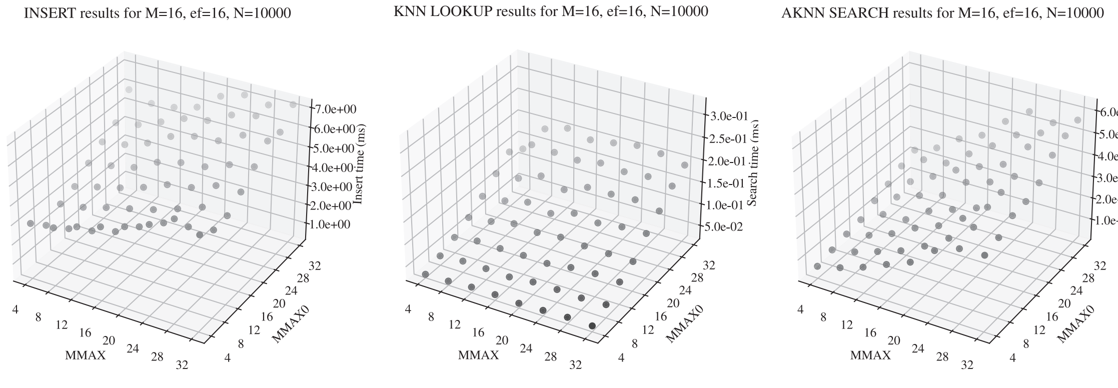
This section first introduces the threat model of APOTHEOSIS and then describes its limitations.

5.1. Threat model

APOTHEOSIS detects similarities between any digital artifacts using SDA. An adversary who is aware of the SDAs in use could manipulate the data to evade detection or cause false positives, for instance by altering byte data to affect similarity scores or disrupting the digest generation and comparison processes (Martín-Pérez et al., 2021). To counter this, our system supports multiple SDA simultaneously, making it difficult for adversaries to exploit vulnerabilities in different algorithms.

The system’s REST API could be susceptible to denial of service attacks if an adversary overloads it with excessive requests or crafted inputs that exhaust resources, leading to unavailability. Mitigation strategies include rate limiting, deploying content delivery networks, load balancing, and continuous monitoring of network and server performance.

An adversary could also attempt to extract sensitive data by querying approximate hashes through the REST API. However, APOTHEOSIS operates on hashes (intermediate representations of artifacts) and does not store original data. Users can control what metadata is returned in search results, minimizing the risk of unwanted data exposure.

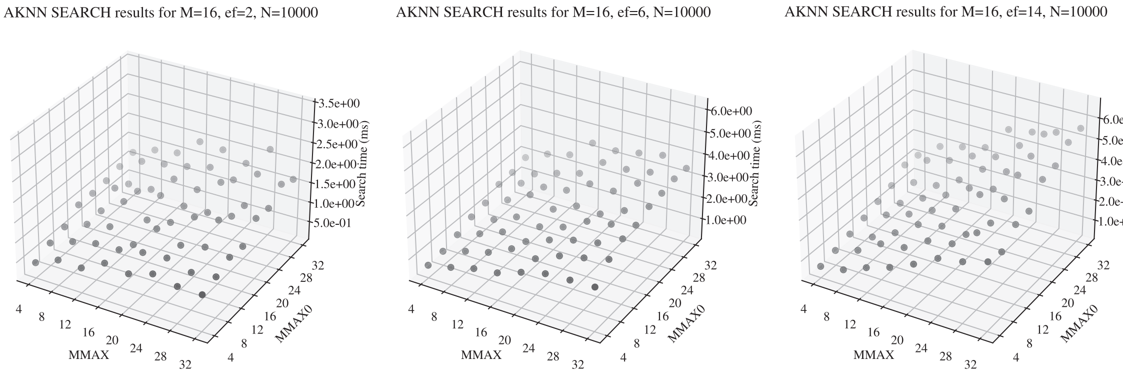


(a) Insert operation

(b) *K*-NN lookup operation

(c) Approximate *K*-NN search operation

Fig. 4. Runtime (in milliseconds) of the insert, K-NN lookup, and approximate K-NN (AKNN) search operations, with $M = ef = 16$ and $N = 10000$.



(a) With $ef = 2$

(b) With $ef = 6$

(c) With $ef = 14$

Fig. 5. Runtime (in milliseconds) of the approximate *K*-NN search operation varying $ef \in \{2, 6, 14\}$, with $N = 10000$ and $M = 16$.

5.2. Threats to validity

We discuss the validity of our work according to construct, internal, and external validity (Cruzes and ben Othmane, 2017).

5.2.1. Construct validity

We conducted controlled experiments using standard evaluation metrics to fine-tune and assess our system. This approach ensures that our experiments measure what is intended, reducing the likelihood of construct validity issues.

5.2.2. Internal validity

APOTHEOSIS relies on third-party libraries to calculate similarity digests and related comparison operations. Any vulnerabilities or bugs in these libraries could affect the effectiveness of the system or be exploited by adversaries.

APOTHEOSIS uses HNSW, which is a graph-based K-ANNS method. While HNSW is a popular choice, there are other K-ANNS methods. Wang et al. (2021) provide a recent survey of graph-based K-ANNS. A more profound and comprehensive evaluation of other K-ANNS methods can help determine which approach is best suited to each problem in the digital forensics domain.

Furthermore, while HNSW is effective, it may not always find the most accurate nearest neighbors due to its approximate nature. Post-processing techniques (such as re-ranking or re-scoring) can improve search precision and be effective to address this limitation.

HNSW can be very memory-intensive, which is a limitation for very large datasets. While compression techniques can reduce its memory usage, they introduce decompression overhead that can degrade performance.

Likewise, constructing the initial HNSW index can be computationally expensive and time-consuming, particularly for large datasets. In this sense, parallel indexing can speed up the construction of the initial HNSW index.

Finally, our experimental results show that the performance of HNSW can be sensitive to parameter settings. Fine-tuning these parameters can be challenging and may require domain-specific knowledge. Therefore, systematic experimentation is necessary to fine-tune HNSW parameters for the specific dataset and query workload. In this sense, automated parameter optimization techniques can also be helpful.

5.2.3. External validity

APOTHEOSIS is specifically designed to work with similarity digests. Therefore, our system cannot be used if the similarity digests of the artifacts cannot be computed. For instance, ssdeep or TLSH cannot generate a digest under certain circumstances (Martín-Pérez et al., 2021).

Our experiments were performed on well-structured datasets with known real-world data. In contrast, real-world forensic data often contains noise, adverse modifications, or missing artifacts, which can affect the performance of similarity search. Therefore, the generalizability of our findings to other datasets, SDA, or contexts may be limited. Future work should include a broader range of datasets and real-world cases to validate the applicability of APOTHEOSIS in diverse situations, including data from real-world forensic cases assess system robustness. Additionally, adversarial testing could help assess resilience against obfuscation techniques used to evade similarity detection.

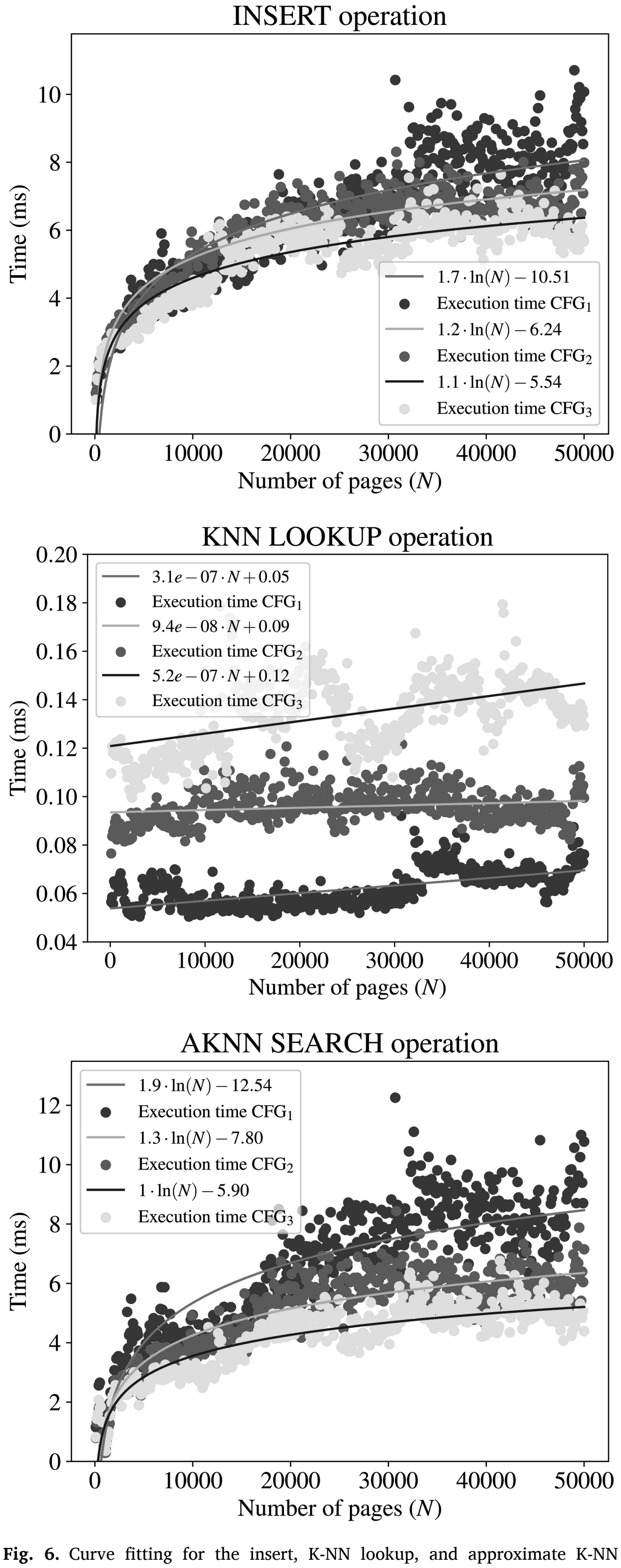


Fig. 6. Curve fitting for the insert, K-NN lookup, and approximate K-NN operation for dataset sizes $N \in [100, 50000]$ for three different APOTHEOSIS configurations (*CFG$_i$*: (*ef*, *M*, *M$_{max}$*, *M$_{max0}$*); specifically, *CFG*$_1$: (4, 4, 16, 16), *CFG*$_2$: (8, 8, 16, 16), and *CFG*$_3$: (8, 16, 16, 32).

6. Related work

In this section, we review tools related to HNSW and previous research on the hash lookup problem and similarity search in digital forensics.

6.1. Tools related to HNSW

HNSW has gained much attention due to its efficiency in approximate nearest neighbor searches. However, existing implementations of HNSW primarily support continuous metrics that rely on continuous vector components, they are not designed to handle deterministic and discrete hashes. As a result, these implementations are not well suited for our use case, requiring custom adaptation to effectively process discrete hash-based representations. The most prominent implementation is HNSWlib (Malkov, 2018), a C++ library with Python3 bindings. While it allows customization of the distance function, it is limited to continuous metrics such as Euclidean distance, inner product, and cosine similarity. These metrics require continuous vector components, which are not suitable for hashes that are deterministic and discrete.

Similarly, Facebook AI Similarity Search (Johnson et al., 2019) is a C++ library with GPU acceleration support, but it also uses vector-based search similarity functions, making it incompatible with hash comparisons. Other HNSW implementations in languages such as Java, Go, and Rust share the same focus on vector search. A recent Python implementation (Bartholomy, 2023) also lacks support for hash-based data. In contrast, APOTHEOSIS is specifically designed for hash-based data. By accommodating hash similarity metrics, our system offers practitioners a specialized and flexible solution to efficiently identify and analyze any digital artifacts.

6.2. Hash lookup and similarity search

Several methods have been developed to improve the efficiency of hash lookup and similarity search, such as n-gram indexing, dynamic programming, and locality-sensitive hashing. Navarro et al. (2001) and Boytsov (2011) provide comprehensive reviews of these techniques, highlighting their applications in approximate string matching and DNA sequence analysis.

In digital forensics, the F2S2 (Winter et al., 2013) approach improves forensic similarity searches by indexing piecewise hash signatures, optimizing SDA such as ssdeep, and significantly reducing search times compared to brute-force approaches. Breitinger et al. (2014b) proposed a divide-and-conquer approach using hierarchical Bloom filters to reduce the search complexity to $\mathcal{O}(\log N)$ on large datasets. Our system uses radix tree for hash lookup, achieving a search time complexity of $\mathcal{O}(k)$, where k is the hash length in bytes. This means that the search time complexity is independent of the number of elements stored, providing efficient performance even with large data sets. Furthermore, unlike our system, Bloom filters, while effective for large-scale set operations, can lead to false positives. In contrast, our approach does not produce false positives but may miss some true matches due to threshold-based filtering. Finally, Liebler et al. (2019) evaluated three hash lookup strategies, measuring runtime and memory consumption, and offering insights for optimizing such systems.

Our work extends these approaches by providing a flexible platform capable of handling multiple SDA and supporting threshold-based searches and AKNNs. For instance, integrating F2S2 with our radix tree could enable fast localization of similar hashes when the SDA generates piecewise hashes, improving approximate K-NN search results. This integration would require some engineering efforts in the insertion and search operations. Similarly, the hash search strategies evaluated by Liebler et al. (2019) could be incorporated as alternatives to our HNSW model.

Also worth mentioning is FRASHER (Göbel et al., 2022), an automated framework for evaluating SDA performance through structured test cases such as fragment detection and efficiency, among others. While this framework focuses on SDA benchmarking, APOTHEOSIS primarily focuses on providing methods for scalable, real-time approximate similarity search on large forensic datasets, with evaluation of SDA and its approximate similarity search on particular forensic problems being a secondary benefit of our system. Thus, our system complements the benchmarking capabilities provided by FRASHER.

7. Conclusions and future work

In this paper, we presented APOTHEOSIS, a flexible and efficient system designed for hash lookup and approximate similarity searches on large forensic datasets using SDA. By combining a radix tree structure with a graph-based HNSW algorithm, APOTHEOSIS offers fast lookups and approximate nearest neighbor searches, optimized for hash-based data. Through two case studies (document duplication detection in a source code plagiarism dataset and system module analysis in memory forensics), we demonstrated its ability to efficiently identify similar artifacts while achieving robust performance in terms of runtime and scalability. Furthermore, our system can also help evaluate the performance of different SDA and approximate similarity searches in particular forensic tasks.

As future work, we plan to investigate and evaluate the accuracy and performance of alternative K-ANNS methods to potentially enhance our system. Furthermore, our ongoing efforts include to further optimize HNSW configuration parameters to improve overall performance. Similarly, we also plan to explore additional use cases and expand the scope of our system to other types of digital artifacts, such as image or video artifacts. Finally, we welcome collaboration with the digital forensics community to gather feedback and insights and enhance the utility and relevance of our system.

Declaration of generative AI and AI-assisted technologies in the writing process

During the preparation of this work, the authors used ChatGPT-4 to improve readability and language. After using this tool/service, the authors reviewed and edited the content as needed and assume full responsibility for the content of the publication.

Acknowledgments

This research was supported in part by grants PID2020-113903RB-I00 (KIT-IA) and PID2023-151467OA-I00 (CRAPER), funded by MICIU/AEI/10.13039/501100011033 and by ERDF/EU, by grant TED2021-131115A-I00 (MIMFA), funded by MICIU/AEI/10.13039/501100011033 and by the European Union NextGenerationEU/PRTR, by grant *Proyecto Estratégico Ciberseguridad EINA UNIZAR*, funded by the Spanish National Cybersecurity Institute (INCIBE) and the European Union NextGenerationEU/PRTR, by grant *Programa de Proyectos Estratégicos de Grupos de Investigación* (DisCo and SID research groups, refs. T21-23R and T42-23R, respectively), funded by the University, Industry and Innovation Department of the Aragonese Government.

References

Bartholomy, 2023. Hierarchical Navigable Small World: a Scalable Nearest Neighbor Search [Online. https://github.com/brtholomy/hnsw. (Accessed 5 June 2023).

Bentley, J.L., 1975. Multidimensional binary search trees used for associative searching. Commun. ACM 18, 509–517.

Boytsov, L., 2011. Indexing methods for approximate dictionary searching: comparative analysis. ACM J. Exp. Algorithmics 16.

Breitinger, F., Guttman, B., McCarrin, M., Roussev, V., White, D., 2014a. Approximate Matching: Definition and Terminology. Techreport NIST Special Publication 800-168. National Institute of Standards and Technology.

Breitinger, F., Rathgeb, C., Baier, H., 2014b. An efficient similarity digests database lookup – a logarithmic divide & conquer approach. J. Dig. Foren. Secur. Law 9, 155–166.

Broder, A., 1997. On the resemblance and containment of documents. In: Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171), pp. 21–29.

Cruzes, D.S., ben Othmane, L., 2017. Empirical Research for Software Security. CRC Press, p. 26 chapter Threats to Validity in Empirical Software Security Research.

Dong, W., Moses, C., Li, K., 2011. Efficient K-nearest neighbor graph construction for generic similarity measures. In: Proceedings of the 20th International Conference on World Wide Web. Association for Computing Machinery, New York, NY, USA, pp. 577–586.

Fernández-Álvarez, P., Rodríguez, R.J., 2022. Extraction and analysis of retrievable memory artifacts from Windows telegram desktop application. Forensic Sci. Int.: Digit. Invest. 40, 301342. Selected Papers of the Ninth Annual DFRWS Europe Conference.

Göbel, T., Uhlig, F., Baier, H., Breitinger, F., 2022. FRASHER – a framework for automated evaluation of similarity hashing. Forensic Sci. Int.: Digit. Invest. 42, 301407. Proceedings of the Twenty-Second Annual DFRWS USA.

Huici, D., Rodríguez, R.J., Mena, E., 2025. Apotheosis: an efficient approximate similarity search system. SoftwareX 29, 102016. https://doi.org/10.1016/j.softx.2024.102016. https://www.sciencedirect.com/science/article/pii/S2352711024003868.

Johansen, G., 2022. Digital Forensics and Incident Response: Incident Response Tools and Techniques for Effective Cyber Threat Response, third ed. Packt Publishing.

Johnson, J., Douze, M., Jégou, H., 2019. Billion-scale similarity search with GPUs. IEEE Transact. Big Data 7, 535–547.

Katz, J., Lindell, Y., 2015. Introduction to Modern Cryptography. CRC Press.

Kornblum, J., 2006. Identifying almost identical files using context triggered piecewise hashing. Digit. Invest. 3, 91–97. The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06).

Leskovec, J., Rajaraman, A., Ullman, J., 2020. Mining of Massive Datasets. Cambridge University Press.

Liebler, L., Schmitt, P., Baier, H., Breitinger, F., 2019. On efficiency of artifact lookup strategies in digital forensics. Digit. Invest. 28, S116–S125.

Ljubovic, V., Pajic, E., 2020. Plagiarism detection in computer programming using feature extraction from ultra-fine-grained repositories. IEEE Access 8, 96505–96514. https://doi.org/10.1109/ACCESS.2020.2996146.

Malkov, Y., 2018. HNSWlib [Online. https://github.com/nmslib/hnswlib. (Accessed 5 June 2023).

Malkov, Y.A., Yashunin, D.A., 2020. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE Trans. Pattern Anal. Mach. Intell. 42, 824–836.

Martín-Pérez, M., 2022. Effectiveness of Similarity Digest Algorithms for Binary Code Similarity in Memory Forensic Analysis. Phdthesis. University of Zaragoza (Spain).

Martín-Pérez, M., Rodríguez, R.J., Breitinger, F., 2021. Bringing order to approximate matching: classification and attacks on similarity digest algorithms. Forensic Sci. Int.: Digit. Invest. 36, 301120.

Microsoft, 2021. Modules [Online. https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/modules. (Accessed 7 October 2022).

Microsoft Docs, 2018. Memory Management [Online. https://docs.microsoft.com/en-us/windows/win32/memory/memory-management. (Accessed 15 February 2020).

Morrison, D.R., 1968. PATRICIA–Practical algorithm to retrieve information coded in alphanumeric. J. ACM 15, 514–534.

Navarro, G., Baeza-Yates, R., Sutineny, E., Tarhioz, J., 2001. Indexing methods for approximate string matching. IEEE Data Eng. Bull. 24, 19–27.

Oliver, J., Cheng, C., Chen, Y., 2013. TLSH – a locality sensitive hash. In: 2013 Fourth Cybercrime and Trustworthy Computing Workshop. IEEE, pp. 7–13.

Pryde, J., Angeles, N., Carinan, S.K., 2018. Dynamic whitelisting using locality sensitive hashing. In: Ganji, M., Rashidi, L., Fung, B.C.M., Wang, C. (Eds.), Trends and Applications in Knowledge Discovery and Data Mining. Springer International Publishing, Cham, pp. 181–185.

Richardson, L., Ruby, S., 2007. RESTful Web Services, first ed. O'Reilly Media, Inc.

Wang, M., Xu, X., Yue, Q., Wang, Y., 2021. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. Proc. VLDB Endow. 14, 1964–1978.

Watts, D.J., Strogatz, S.H., 1998. Collective dynamics of 'small-world' networks. Nature 393, 440–442.

Webster, A.F., Tavares, S.E., 1986. On the design of S-boxes. In: Advances in Cryptology — CRYPTO '85 Proceedings. Springer Berlin Heidelberg, pp. 523–534.

Winter, C., Schneider, M., Yannikos, Y., 2013. F2S2: fast forensic similarity search through indexing piecewise hash signatures. Digit. Invest. 10, 361–371.

Yianilos, P.N., 1993. Data structures and algorithms for nearest neighbor search in general metric spaces. In: Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, USA, pp. 311–321.