

Impact of Distances in an Anomaly Detection Context for Time Series in Software Testing

Kevin Gerardo Polo Ruiz(✉), Alexandre Claudio Botazzo Delbem, and Paulo Sergio Lopes de Souza

University of Sao Paulo (ICMC/USP), Sao Paulo, Brazil
kpruiz11@gmail.com, {acbd,pssouza}@icmc.usp.br

Abstract. Software Testing is a critical stage in Software Development projects, which increases their quality at the cost of an increasing budget. This cost grows as the System Under Test (SUT) has poor testability. Data streaming systems or web servers usually present poor testability because they have constant input rates that often hide faults during testing. For such systems, it is costly to build test cases and verify whether the output is correct for each particular input. Tricorder proposes a methodology for anomaly detection without system requirements, taking into account the unsupervised learning provided by Damicore's methodology. This paper evaluates the behavior of distance measures for time series data within the frameworks of unsupervised machine learning carried out by Damicore, which is applied to software anomaly detection. The study systematically assesses the impact of each distance measure on the resulting accuracy within Damicore by conducting experiments across multiple benchmarks. Our findings reveal that the Levenshtein distance significantly outperforms DTW, NCD, FFT, and Hamming. Conversely, Hamming distance demonstrates the poorest performance. Levenshtein distance, on the other hand, offers an excellent balance between execution time and accuracy, providing an optimal trade-off for efficient and effective fault detection. The findings provide valuable insights into the effectiveness of these measures in enhancing fault detection accuracy, which could be extended to other domains involving anomaly detection of time-series data.

Keywords: Time Series · Anomaly Detection · Agglomerative Clustering · Fault Detection · Software Testing

1 Introduction

In the current age of the Fourth Industrial Revolution, where almost everything is digitally connected [23], new systems tend to be more complex, increasing the presence of integrity and security issues, with possible economic and personal losses. Validation, Verification, and Test (VV&T) activities are required

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025
O. Gervasi et al. (Eds.): ICCSA 2025, LNCS 15649, pp. 18–34, 2025.
https://doi.org/10.1007/978-3-031-96997-3_2

to improve software quality, bringing high costs up to 40% of the whole budget for projects [28]. There is a pressing need for effective testing methodologies applied to software systems with poor testability, which may arise from factors that hinder the generation of test cases, such as legacy system constraints, incomplete or missing documentation, lack of clearly defined requirements, limited access to system internals, or challenges in replicating real-world scenarios [31].

Tricorder [17] is a novel software testing methodology to alert to possible faults, developed due to the need to execute testing in a maintenance scenario. It is a complementary testing methodology used after usual techniques, such as unit and integration testing, have already been applied. Tricorder was conceived thinking in systems with poor testability [24,29,31], like systems that process streaming data or requests on web servers. These systems have a constant rate of inputs that often prevent faults from being detected during preliminary tests. It is highly costly to build test cases and verify that each output is correct for its particular input [17].

Tricorder detects anomalies in the utilization of computing system resources by comparing execution profiles of a modified system version against a well-tested reference version. It leverages Damicore [22], a methodology focused on unsupervised learning. It applies clustering techniques to automatically identify behavioral deviations in the system through anomaly detection principles applied to time-series data. Damicore, as a core module of Tricorder, operates through three sequentially dependent stages: distance matrix generation, tree simplification, and community detection, ensuring a systematic and practical analysis of system behavior.

The construction of the distance matrix by Damicore is a fundamental stage since its representativeness influences the achievements of other steps. This paper evaluates different distance metrics from the literature on time series fault detection in software testing when using Damicore applied to the Tricorder context. Challenging test scenarios are proposed to check their limits in capturing significant differences from software profiles. Most scenarios relate to software maintenance, which requires identifying the presence and absence of faults, i.e., resource consumption anomalies. The number of false positives or negatives is a way to estimate the performance of such a tool. A balance is expected for practical applications, where the cost of false positives (unnecessary bug fixes) and false negatives (missed faults) can be substantial.

This work presents a systematic approach for evaluating the performance of our model. Although this work focuses on exploring and evaluating our methodology within a software testing context, its principles and findings could be extended to other domains involving time-series anomaly detection. For instance, fields such as health monitoring [10] and financial fraud detection [19] could significantly benefit from the insights and results presented in this study.

The remainder of the paper is as follows. Section 2 introduces the distance metrics in the literature that will also be part of this paper. Section 3 presents Tricorder and its relationship with measurements of software profiles used for fault detection based on computer resource usage. Section 4 describes the

methodology for evaluating the performance of Tricorder according to each of the metrics, which involves a set of applications and workload scenarios. Section 5 and 6 present results and a detailed analysis. Finally, Sect. 7 highlights fundamental aspects of the proposed study and concludes the paper.

2 Related Work

The related work described in this Section addresses the construction of the distance matrix, which is the focus of our research.

The significance of Dynamic Time Warping (DTW) in time series analysis is well-documented. DTW aligns sequences by stretching or compressing them, which allows for identifying patterns even when there are temporal shifts. This flexibility makes DTW particularly useful for various applications in anomaly detection, being recognized as a robust and stable resource for handling time series data, outperforming Euclidean distance in some contexts using agglomerative clustering [11].

Another essential technique in the literature is the Fast Fourier Transform (FFT), which is frequently used for its efficiency in transforming time series data into the frequency domain. By doing so, FFT simplifies the identification of periodic components and anomalies within the data. This technique's efficacy is extensively discussed in [4], where FFT's role in processing and analyzing time series data is thoroughly explored.

In addition to these traditional methods, our project also investigates utilizing binary information extracted from logs, applying various distance metrics such as the default option in Damicore, Normalized Compression Distance (NCD) [5]. NCD is advantageous because it measures the similarity between sequences based on their compressibility, providing a unique perspective on anomaly detection that is particularly effective for any kind of data, including log data.

Levenshtein distance is another known metric that is applied to convert time series data into string representations. Levenshtein distance for time series data presents promising results for clustering [27], outperforming DTW and Euclidean in different datasets and demonstrating its capability to effectively calculate distances and identify anomalies by assessing the edits required to transform one string into another.

Hamming distance is also found in the literature, especially for time series data represented as fixed-length binary strings. This metric is helpful for its simplicity and effectiveness in counting the differing bits between two sequences. It is particularly well-suited for identifying anomalies in binary data, providing a straightforward method to detect discrepancies and faults in time series.

By leveraging these diverse distance metrics, we aim to enhance Damicore's robustness and accuracy for the detection of anomalies in the utilization of computing resources in the Tricorder context. Each technique brings its strengths to the table, offering unique insights and capabilities that contribute to the comprehensive detection of abnormal behaviors within our datasets.

3 Tricorder and Damicore

Tricorder [17] is a testing methodology designed to detect potential faults during software maintenance by identifying anomalous behavior in computing systems without relying on their internal structure or code. This approach is efficient for systems with low testability, where defining a test oracle is challenging due to the lack of clearly defined expected outputs. Inspired by regression testing, Tricorder compares the behavior of a modified system version against a previously well-tested version, referred to as the "reference," to identify changes in system performance.

As illustrated in Fig. 1a, after a code modification - often made during software maintenance - the behaviors of the reference version and the system version under test are monitored. During multiple executions, resource usage is captured at fixed intervals, generating time-series data used to construct a performance profile that represents the system's functionality. Deviations in these profiles from the expected norms of the reference version indicate a system's unexpected behavior and, consequently, the existence of possible software faults.

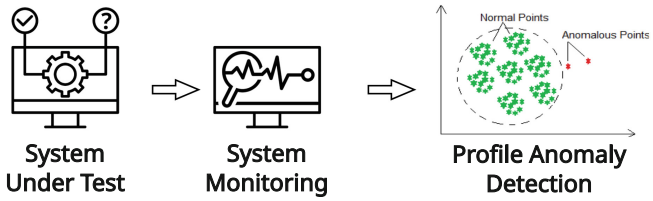
Tricorder has been evaluated across various contexts for fault detection, including applications in multiple domains [8] and machine learning systems [3]. Additionally, the methodology accounts for various system resources during execution, depending on the application type. Previous experiments demonstrated that bugs can significantly impact system resources such as CPU, RAM, disk I/O, and GPU [3,8,17].

The anomaly detection phase of Tricorder leverages Damicore [22] to analyze log data, grouping execution traces and identifying anomalous executions that deviate from expected behavior. The Damicore methodology, based on unsupervised learning, is capable of identifying patterns in any data by applying a hierarchical agglomerative clustering that uses modularity optimization to group execution traces from both the reference version and the version under test, enabling effective anomaly detection across diverse datasets. Damicore clusters executions from both versions, and if a cluster consists only of executions from the new version, it indicates the presence of a fault. Illustrated in Fig. 1b, Damicore receives as input a Profile Dataset and has three key steps:

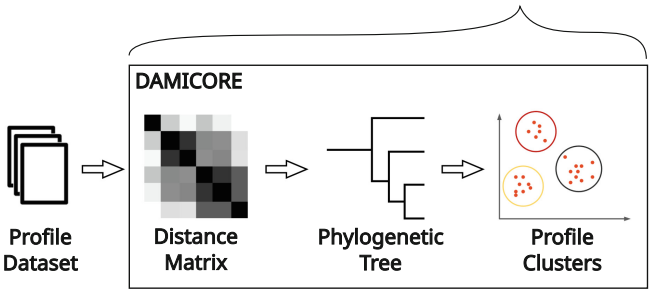
- **Distance Matrix**: A matrix is generated using distance metrics applied between execution logs, with Normalized Compression Distance (NCD) [5] as the default measure. NCD is a compression-based metric derived from Kolmogorov complexity, enabling the measurement of distances between arbitrary data types. This capability ensures Damicore's versatility in handling diverse datasets, making it adaptable to various anomaly detection scenarios. Each element represents the distance between logs from two executions, and the matrix is iteratively updated with new version logs.
- **Phylogenetic Tree**: A hierarchical structure representing relationships between execution traces based on their similarities. In agglomerative clustering, this tree - also known as a dendrogram - illustrates how data points merge, with closer branches indicating higher similarity. In our approach, the

complete graph is reduced to a simplified tree using the Neighbor-Joining algorithm [21]. It provides a visual representation of the clustering process, where closer branches indicate higher similarity, making it helpful in understanding patterns in unsupervised learning tasks.
– **Profile Clusters**: Communities are identified within the tree using the Fast Newman algorithm [18].

Despite its utility, a gap remains in analyzing the impact of time-series distance measures for anomaly detection in the context of fault identification, as discussed in Sect. 2.



(a) Tricorder: Methodology



(b) DAMICORE: Methodology

Fig. 1. Tricorder and Damicore methodologies.

4 Methodology

In this Section, we outline the methodology applied during the execution of this paper. We begin by defining the scope of applications included in the analysis, most of which have been previously tested using Tricorder [3,17]. These applications were selected, prioritizing highly evaluated and well-documented open-source projects to ensure the reliability and relevance of the analysis. While we recognize the inherent limitation of testing every possible application, the selected set of applications represents a diverse range of characteristics common to many, including streaming services, machine learning projects, and database management systems.

Each application was evaluated using a set of test cases, referred to as workloads in this study, designed to assess key application features comprehensively. To analyze fault detection effectiveness, we tested multiple versions of each system, each containing a different inserted fault. Additionally, to assess potential over-sensitivity, we included a fault-free version, referred to as *CONTROL*, where no anomalies are expected to be detected. The faults were either intentionally introduced by researchers or replicated from real-world issues documented by the community on GitHub [3,32]. This approach ensures that the tests reflect real-world behavior while minimizing selection bias.

4.1 Network Telemetry

Network Telemetry is a streaming application that was initially documented in Tricorder's proposal [16]. In this benchmark, we use data logged already published [15], which contained logs of CPU, RAM, and Disk IO usage during multiple executions of this application. The outline of workloads and faults is as follows:

- Workloads: Defined by the format and size of files the application processes.
  - Format: Binary (BIN), Comma-Separated Values (CSV), and both formats (BINSCV).
  - Size: LOW, MEDIUM, and HIGH sized files.
- Faults
  - BCLEAN: a memory leak.
  - MONO: a parallelism problem.
  - SLEEP: null CPU usage.
  - SWAP: swap memory addresses.
  - INFINITE: high CPU usage.
  - NOBREAK: wrong execution flow.
  - UNLOCK: concurrency desync.

4.2 Voice Cloning

A machine learning application [6,7] that synthesizes audio by cloning the voice from a provided audio with a text. We execute the monitoring stage of Tricorder in this benchmark, according to the workloads and faults defined in previous experiments [3].

- Workloads: Defined by the length of provided audio and text;
  - Audio: Short and Long duration audio.
  - Text: Short, Medium, and Long sized text.
- Faults;
  - API: Missing or wrong API calls.
  - CONC: Problems of concurrence and synchronization.
  - LOGIC: Wrong logic or flow executions.
  - MEMORY: Memory usage problems.
  - MODEL: Misconfigured model.
  - PROCESS: Wrong initialization of variables or methods.
  - TRAIN: Wrong usage of the training set.

4.3 Bert

This benchmark implements the model proposed for solving natural language processing problems [6,7]. We use the data processed by Braga [3], already published [2].

- Workloads: Defined by the Datasets used for training: MRPC, CoLA, MNLI, XNLI, SQuaDv1, and SQuaDv2.
- Faults: Similar to the Voice Cloning benchmark: API, CONC, LOGIC, MEMORY, MODEL, PROCESS, TRAIN.

4.4 Chess Alpha Zero

This benchmark implements a Chess version of the AlphaGo Zero model [20,26], which learns to play by predicting the next move. We use the data processed by Braga [2,3] for our analysis.

- Workloads: Defined by the dataset used for training: OPT, OPT_STEP, EVAL, SELF, and SUP_LEARN.
- Faults: Similar to the Voice Cloning benchmark: API, CONC, LOGIC, MEMORY, MODEL, PROCESS, TRAIN.

4.5 Database

PostgreSQL [9], one of the most used relational database management systems, was executed in the monitoring stage of Tricorder. We defined a new set of faults and workloads.

- Workloads: Defined by the IO operation and the execution load;
  - IO Operation: READ, WRITE, and MIXED.
  - Load: LOW and HIGH size loads.
- Faults
  - DISKSORT: Forces sort operations on disk.
  - HASHJOIN: Forces join using the Hash algorithm.
  - MEMJOIN: Forces join on memory.
  - MONO: Disabling concurrency.
  - INFINITE: High CPU usage.
  - SLEEP: Low CPU usage.

In the context of our research paper, a correct selection of distances is essential to ensure the robustness and generalization of the proposed approach.

In this work, we modify Damicore to evaluate the effectiveness of various distance measures for time series, as introduced in Sect. 2. These modifications aim to address the identified gaps in time-series distance analysis for anomaly detection. Below, we detail the selected distance measures and their relevance to the study:

- Normalized Compression Distance: The default distance metric implemented in Damicore, NCD, measures the similarity between objects based on their compressed sizes. Its versatility lies in being independent of the nature of the data, making it suitable for a wide range of applications.
- Dynamic Time Warping: particularly effective for time-series analysis as it accounts for temporal distortions. We utilize an implementation capable of handling multidimensional time series (See [14,25]).
- Fast Fourier Transform: Employed to analyze the frequency domain characteristics of time-series data. For this study, we use the multidimensional time-series implementation available in SciPy [30].
- Levenshtein: Measures the edit distance between sequences, offering insight into the structural changes in the data [1].
- Hamming: Similar to Levenshtein, Hamming distance evaluates the differences between sequences of equal length, focusing on direct substitutions [1].

By integrating these diverse distance measures, we aim to evaluate their performance comprehensively and identify the most effective approaches for anomaly detection in fault detection contexts.

As presented in Sect. 1, we prioritize accuracy as our primary performance metric for anomaly detection, as it is particularly relevant to our context. Accuracy is defined as the proportion of correctly identified executions where a faulted version triggered an alert and where a non-faulted version did not, relative to the total number of executed workload-version combinations for each application. This metric provides a clear and straightforward measure of detection performance, minimizing unnecessary alerts while maximizing the identification of actual faults. Doing so enhances the robustness and efficiency of the software maintenance process. Other metrics, such as the F1-score, are not considered, as our primary focus is on fault detection rather than balancing precision and recall.

One significant issue identified was the presence of negative branches when employing the neighbor-joining method for Dynamic Time Warping (DTW). This phenomenon was observed particularly with our multidimensional data, where DTW failed to adhere to the triangle inequality. In this context, we applied the Fast Newman method with the inverse of the edge length, which precipitated negative weights. This choice led to the failure of the Fast Newman algorithm. As suggested by [12], a viable solution is to assign a zero value to the negative branches, thereby avoiding negative weights and preserving the network topology. However, this adjustment resulted in zero-length branches, introducing indeterminate weights. To address this, we explored three weight generation strategies: the default inverse option, normalizing lengths for inverse calculation, and applying a constant edge weight of 1, effectively creating an unweighted network (See Table 1).

Finally, we evaluate the influence of these distances on the accuracy of fault detection, where higher accuracy corresponds to superior outcomes. To assess the effectiveness of each distance metric, we will conduct a statistical analysis of their performance. We will generate 100 bootstrapped samples, which will

Table 1. Weight Generation

Name	Weight Value
None	$W = 1$
Inverse	$W = \frac{1}{L}$
Normalize	$W = \frac{1}{1+\Delta+\frac{L-\mu}{\sigma}}$

be calculated in the 25%, 50%, and 75% quartiles, also known as the first, second, and third quartiles. The differences between subsets will be analyzed for each statistical measure, and we will compare the 20% central portion of the sample relative to zero. Based on these comparisons, we will determine whether one strategy offers better, worse, or statistically equivalent accuracy in detecting anomalies. Additionally, we will examine the resource costs associated with this impact, focusing mainly on execution time to identify which distances demonstrate reduced execution duration. Ultimately, we will explore our findings to clarify the underlying explanations for these results.

5 Results

To provide an initial overview of the performance associated with each distance metric, we present the distribution of accuracy values in Fig. 2. This figure depicts the results using boxplots, enabling visualization of key statistical measures such as medians, quartiles, and the spread of the accuracy obtained in all distance metrics evaluated.

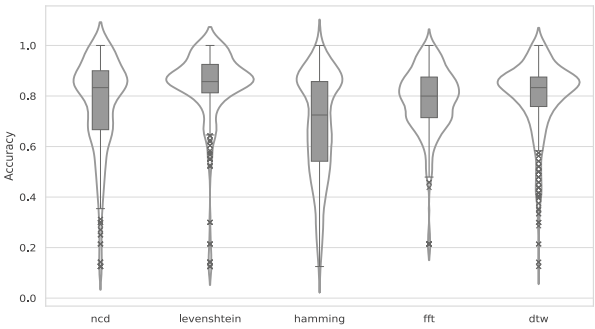


Fig. 2. Accuracy by distance.

Since none of the datasets follow a standard distribution, traditional statistical tests are not applicable. Instead, we applied the bootstrapping analysis detailed in the methodology to examine the quartiles of each distribution. The results of this analysis are summarized in Table 2.

Table 2. Results by distance

	ncd(%)	levenshtein	hamming	fft	dtw
Q_1	66.99 ± 0.76	$\mathbf{81.69 \pm 0.69}$	53.74 ± 1.06	71.26 ± 0.27	75.68 ± 1.1
Q_2	83.17 ± 0.52	$\mathbf{85.75 \pm 0.14}$	71.99 ± 1.32	79.65 ± 0.53	83.87 ± 0.66
Q_3	89.35 ± 0.93	$\mathbf{92.18 \pm 0.77}$	85.89 ± 0.54	86.77 ± 0.86	87.5 ± 0.0

As an example, Fig. 3 displays the bootstrapped distributions of the third quartile computed from 100 resampled datasets for each distance metric. The visualization clearly shows that the Levenshtein distance consistently yields higher third-quartile accuracy scores, while the Hamming distance demonstrates notably lower performance in comparison.

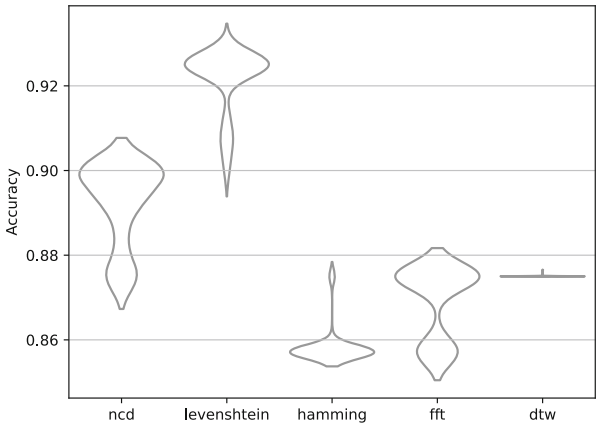


Fig. 3. Q3(75% Quantile) accuracy distribution by distance.

After replicating the analysis across all combinations of distance metrics and identifying negligible differences between bootstrapped datasets, we performed the statistical comparisons outlined in the methodology. Based on these results, we established the following order of accuracy, progressing from lower to higher, within each quartile:

- Q_1: {hamming} < {ncd} < {fft} < {dtw} < {levenshtein}
- Q_2: {hamming} < {fft} < {ncd} < {dtw} < {levenshtein}
- Q_3: {hamming} < {fft} < {dtw} < {ncd} < {levenshtein}

From the evaluation of each quartile, we conclude that the Hamming distance consistently presents the lowest accuracy in every quartile, while the Levenshtein distance shows the highest accuracy. The other distances, namely NCD, FFT, and DTW, fall between these extremes, with their relative performance varying across quartiles.

This consistent pattern suggests that the Levenshtein distance is the most influential metric for our fault detection approach, providing the highest accuracy across all quartiles. Conversely, the Hamming distance is the least effective, yielding the lowest accuracy. The performance of NCD, FFT, and DTW varies, indicating that their effectiveness may depend on the specific characteristics of the data in each quartile. This insight helps guide the selection of distance metrics for optimizing fault detection accuracy in our project.

In previous works, it was expected that Dynamic Time Warping (DTW) would present better results for aligning and comparing time series due to its ability to handle time series with varying lengths and non-linear distortions. However, our results indicate that Levenshtein distance can be more effective for anomaly detection in specific contexts. This effectiveness is likely because anomalies often manifest as local deviations from the norm, and Levenshtein distance excels at identifying such changes.

Table 3. Time Execution in seconds by distance

	hamming	levenshtein	fft	dtw	ncd
1. Matrix	$\mathbf{02.438 \pm 0.030}$	40.833 ± 0.622	44.610 ± 0.619	49.867 ± 0.404	94.313 ± 1.817
2. Others	19.413 ± 0.080	19.035 ± 0.091	19.173 ± 0.342	18.979 ± 0.078	19.201 ± 0.077
Total	$\mathbf{21.851 \pm 0.080}$	59.868 ± 0.597	63.784 ± 0.602	68.846 ± 0.445	113.514 ± 1.806

Table 3 presents the response times for community detection using different distance measures, splitting into two stages: "Matrix Distance" and "Others." The total response time is the sum of these two stages. The measures compared include Hamming, Levenshtein, FFT, DTW, and NCD. In the Matrix Distance, we can see:

- Hamming distance is the fastest in computing the matrix distance. This result suggests it is highly efficient for this stage.
- Levenshtein takes a more computationally intensive process for calculating the matrix distance.
- FFT, slightly longer than Levenshtein, highlighting the additional complexity involved in FFT calculations.
- DTW shows a further increase in computation time due to the dynamic programming nature of DTW.
- NCD is the most time-consuming, reflecting its high computational demands for matrix distance calculation.

The "Others" category response times are relatively consistent across all measures. This consistency suggests that the choice of distance measure does not significantly impact the tasks in this stage. All measures' standard deviations are relatively small, indicating stable and consistent response times across different runs. However, NCD shows higher variability in the matrix distance computation, suggesting potential fluctuations in its performance.

6 Discussion

To analyze the trade-off between accuracy and execution time, we present the execution load and accuracy in Fig. 4.

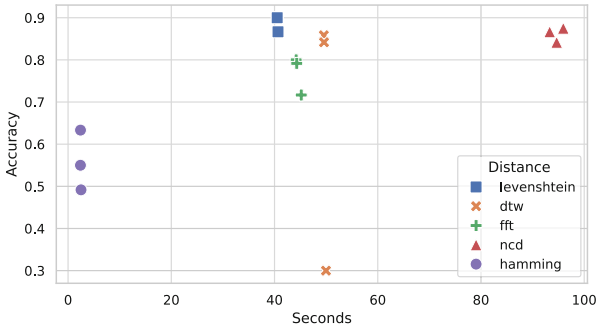


Fig. 4. Time execution vs accuracy.

Our analysis reveals the following insights:

- Efficiency: Hamming distance presented the best results regarding computational efficiency, making it ideal for scenarios where speed is crucial. However, it presents the worst results in accuracy for anomaly detection, which is an essential consideration for applications where accurately detecting anomalies is critical.
- Complexity: Measures like NCD, despite their potentially more prosperous information, come with higher computational costs and variability, which could be a trade-off in time-sensitive applications.
- Balanced Choices: Levenshtein, FFT, and DTW offer moderate trade-offs between computational complexity and richness of information, suitable for applications where moderate computation time is acceptable.

To understand the behavior of each distance, we show the distributions of distances between entities. We separated such entities' data into two groups: a set with reference behavior and a set with anomalous behavior. After this, the distance matrix from combinations of the data is categorized into the following groups:

- *1. Ref vs Ref*: This represents the distance between two reference objects. Since these objects are expected to exhibit normal, non-anomalous behavior, the distance between them should be relatively low. This result indicates that their time series patterns are similar and belong to the same cluster in a typical clustering algorithm used for anomaly detection.
- *2. Def vs Def*: This represents the distance between two anomalous objects. Despite being anomalies, these objects are expected to have similar patterns

of deviation from the norm, leading to a lower distance between them. Identifying this group helps understand common features among different types of anomalies, which can be crucial for refining anomaly detection algorithms.
– *3. Def vs Ref*: This represents the distance between one reference object and one anomalous object. As the behavior of anomalous objects deviates significantly from the reference objects, the distance between them is expected to be higher. This distinction is critical in anomaly detection, as it helps to effectively separate normal time series patterns from those indicative of anomalies.

This categorization helps identify and understand the distinct behavioral patterns between reference and anomalous entities.

In Fig. 5, we show the distributions grouped by the chosen benchmarks. We show slight separability between the group *3. Def vs Ref*, and the other two groups, following the expected behavior. Although the individual datasets yield varying results, the boxplots reveal a consistent trend in the relative ordering among the three data groups.

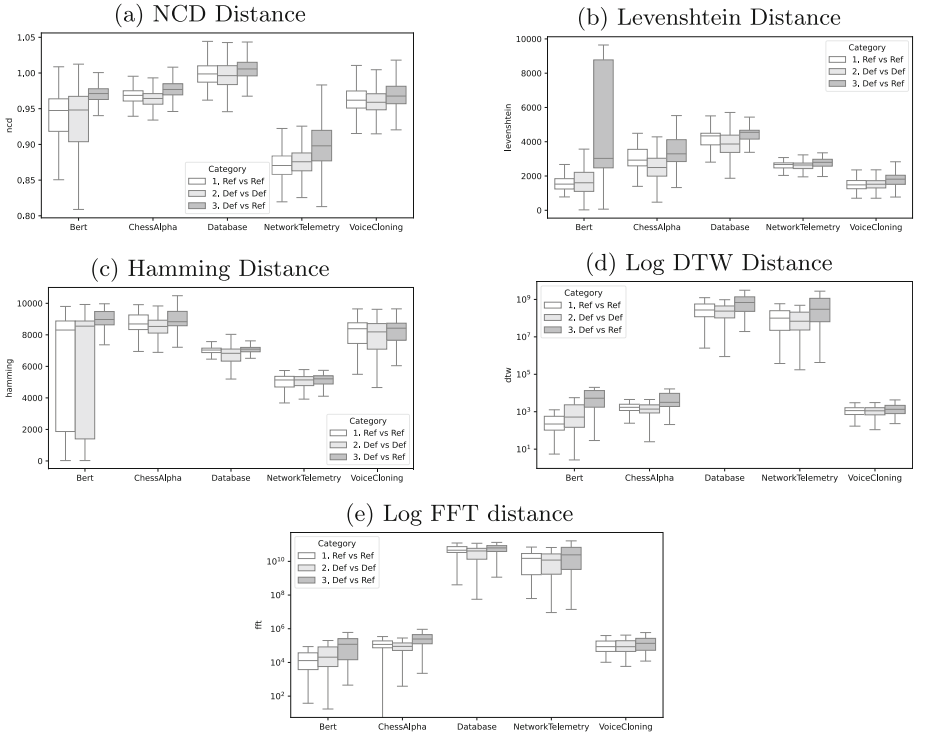


Fig. 5. Distribution for all distances.

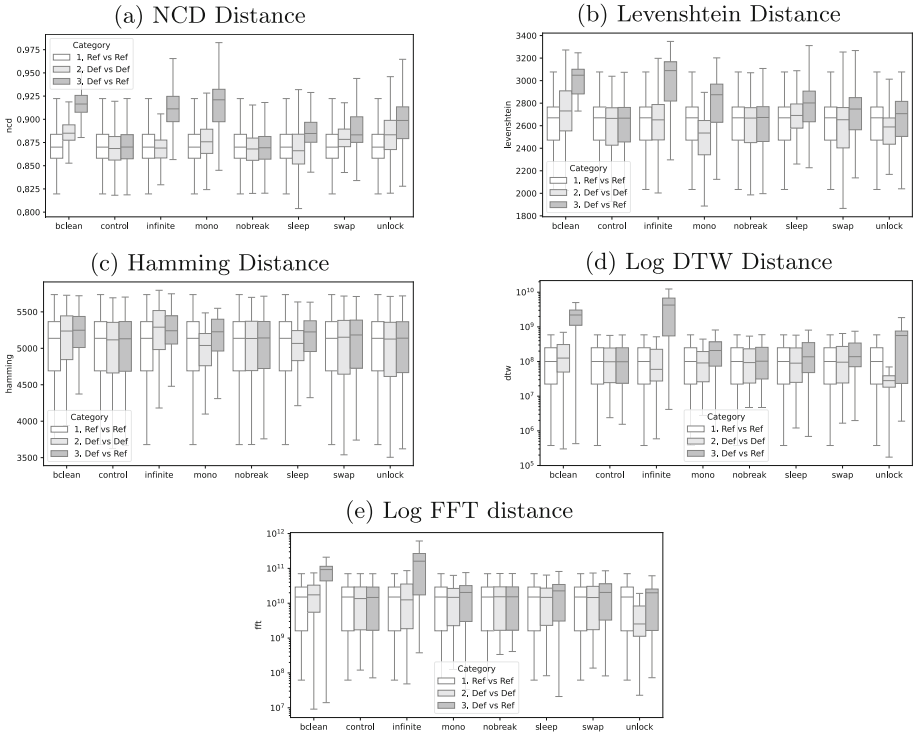


Fig. 6. NetworkTelemetry: Distribution for distances.

To deepen this analysis, we show the distributions of distances in the context of one benchmark, Network Telemetry, the first executed benchmark [17]. As seen in Fig. 6, all distances are visibly capable of detecting differences between objects, with more complications for the Hamming distance. Another notable result is the magnitude of the distributions of distances. Distances with higher accuracy have narrower relative ranges, while those with lower accuracy, such as DTW and FFT, have broader relative ranges. NCD and Levenshtein distances exhibit higher accuracy in distinguishing between reference and anomalous objects. A similar result is presented in the context of voice analysis [13]. Levenshtein outperforms DTW and Euclidean for detecting accents in voice files. This result suggests that these measures are more effective at capturing the underlying differences in behavior, potentially due to their ability to account for more nuanced variations in the data compared to other distance metrics, in contrast to others with higher sensitivity.

7 Conclusion and Future Work

The application of Damicore analysis within the Tricorder context to compare system behavior between well-tested and faulted versions demonstrates that the

Levenshtein distance achieves the highest accuracy for generating a distance matrix while maintaining reasonable execution time. This method outperforms alternatives such as DTW, NCD, FFT, and Hamming distance, paving the way for optimized testing activities within the academic community.

Although this study focuses on fault detection in software systems, the contributions introduced here have the potential to be extended to other fields involving time-series anomaly detection. The principles of this approach, including the application of various distance measures and the systematic evaluation methodology, can benefit domains such as health monitoring and financial fraud detection.

For future work, while our study concentrated on the initial stage of Damicore, subsequent research will explore its later stages, including tree simplification and community detection. By investigating these components and proposing new enhancements, we aim to refine our methodology, leading to more robust, efficient, and effective fault detection processes.

A potential threat to the validity of this study is the reduced diversity of applications and testing coverage. This constraint restricts a broader assessment of Tricorder's flexibility across different functional contexts. To mitigate this threat, we selected five applications from three distinct and widely used domains in the software industry: Streaming, Database, and Machine Learning. Additionally, workloads were designed to maximize feature coverage despite the challenges of generating test cases posed by data variability and volume. While Tricorder and Damicore are application-agnostic, their stability in distinct environments requires further investigation. Future work will expand our experiences across multiple programming languages and frameworks to further validate Tricorder's applicability.

Acknowledgments. The authors would like to thank CAPES (PROEX), CNPq (308445/2021-0) and FAPESP (2013/07375-0, 2019/06937-0, 2019/26702-8) for their financial support which made this research possible.

Disclosure of Interests. The authors declare that they have no conflicts of interest related to this research.

References

1. ax Bachmann: Levenshtein python c extension module. https://github.com/rapidfuzz/Levenshtein (2023)
2. Braga, D.: Tricorder machine learning: Dataset. https://github.com/diegobraga92/TricorderMachineLearning (2021)
3. Braga, D.: Assessing the use of performance data clustering for supporting software testing in the machine learning domain (in Portuguese). Master dissertation in computer science and computational mathematics, University of Sao Paulo, Sao Carlos (2022)
4. Brigham, E.O., Morrow, R.E.: The fast fourier transform. IEEE Spectr. **4**(12), 63–70 (1967). https://doi.org/10.1109/MSPEC.1967.5217220

5. Cilibrasi, R., Vitanyi, P.: Clustering by compression. IEEE Trans. Inf. Theory **51**(4), 1523–1545 (2005). https://doi.org/10.1109/TIT.2005.844059
6. Devlin, J.: BERT. https://github.com/google-research/bert (2022)
7. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. CoRR arxiv:abs/1810.04805 (2018). http://arxiv.org/abs/1810.04805
8. Duraes, T.d.J.O.: Evaluating the use of performance profiles applied in software testing in different application domains (in Portuguese). Master dissertation in computer science and computational mathematics, University of Sao Paulo, Sao Carlos (2022)
9. Group, P.G.D.: PostgreSQL database management system. https://github.com/postgres/postgres (2024)
10. Khanizadeh, F., Ettefaghian, A., Wilson, G., Shirazibeheshti, A., Radwan, T., Luca, C.: Smart data-driven medical decisions through collective and individual anomaly detection in healthcare time series. Int. J. Med. Informatics **194**, 105696 (2025)
11. Kotas, M., Leski, J., Moroń, T., Guzmán, J.G.: Hierarchical agglomerative clustering of time-warped series. In: Gruca, A., Czachórski, T., Harezlak, K., Kozielski, S., Piotrowska, A. (eds.) Man-Machine Interactions 5, pp. 207–216. Springer International Publishing, Cham (2018)
12. Kuhner, M.K., Felsenstein, J.: A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates. Mol. Biol. Evol. **11**(3), 459–468 (1994)
13. Lind-Combs, H.C., Bent, T., Holt, R.F., Clopper, C.G., Brown, E.: Comparing levenshtein distance and dynamic time warping in predicting listeners' judgments of accent distance. Speech Commun. **155**, 102987 (2023)
14. Meert, W., Hendrickx, K., Van Craenendonck, T., Robberechts, P., Blockeel, H., Davis, J.: Dtaidistance (2020)
15. Montes, V.: Tricorder method experimental data (2019). https://doi.org/10.17632/7HC8PK6HTW.1, https://data.mendeley.com/datasets/7hc8pk6htw/1
16. Montes, V.S.: Networktelemetry. https://gitlab.com/vmontes/networktelemetry.git (2020)
17. Montes, V.S.: Software fault detection using clustering of performance profiles (in Portuguese). Master dissertation in computer science and computational mathematics, University of Sao Paulo, Sao Carlos (2020). https://doi.org/10.11606/D.55.2020.tde-04022020-093604, https://www.teses.usp.br/teses/disponiveis/55/55134/tde-04022020-093604/
18. Newman, M.E.J.: Fast algorithm for detecting community structure in networks. Phys. Rev. E **69**(6) (2004). https://doi.org/10.1103/physreve.69.066133
19. Ngai, E., Hu, Y., Wong, Y., Chen, Y., Sun, X.: The application of data mining techniques in financial fraud detection: a classification framework and an academic review of literature. Decis. Support Syst. **50**(3), 559–569 (2011)
20. Pang, M.: Chess alpha zero. https://github.com/Zeta36/chess-alpha-zero (2022)
21. Saitou, N., Nei, M.: The neighbor-joining method: a new method for reconstructing phylogenetic trees. Mol. Biol. Evol. **4**(4), 406–425 (1987). https://doi.org/10.1093/oxfordjournals.molbev.a040454
22. Sanches, A., Cardoso, J.M., Delbem, A.C.: Identifying merge-beneficial software kernels for hardware implementation. In: 2011 International Conference on Reconfigurable Computing and FPGAs, pp. 74–79 (2011). https://doi.org/10.1109/ReConFig.2011.51
23. Sarker, I.H.: Machine learning: algorithms, real-world applications and research directions. SN Comput. Sci. **2**(3), 160 (2021)

24. Sharma, R., Saha, A.: A systematic review of software testability measurement techniques. In: 2018 International Conference on Computing, Power and Communication Technologies (GUCON), pp. 299–303 (2018). https://doi.org/10.1109/GUCON.2018.8675006
25. Shokoohi-Yekta, M., Hu, B., Jin, H., Wang, J., Keogh, E.: Generalizing DTW to the multi-dimensional case requires an adaptive approach. Data Min. Knowl. Disc. **31**(1), 1–31 (2016)
26. Silver, D., et al.: Mastering the game of go without human knowledge. Nature **550**(7676), 354–359 (2017)
27. Tamura, K., Ichimura, T.: Clustering of time series using hybrid symbolic aggregate approximation. In: 2017 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1–8 (2017). https://doi.org/10.1109/SSCI.2017.8280846
28. Valle-Gomez, K.J., Delgado-Perez, P., Medina-Bulo, I., Magallanes-Fernández, J.: Software testing: cost reduction in industry 4.0. In: 2019 IEEE/ACM 14th International Workshop on Automation of Software Test (AST), pp. 69–70 (2019). https://doi.org/10.1109/AST.2019.00018
29. Vianna, A., Kamei, F.K., Gama, K., Zimmerle, C., Neto, J.A.: A grey literature review on data stream processing applications testing. J. Syst. Softw. **203**, 111744 (2023)
30. Virtanen, P., et al.: SciPy 1.0: fundamental algorithms for scientific computing in python. Nat. Methods **17**, 261–272 (2020). https://doi.org/10.1038/s41592-019-0686-2
31. Voas, J., Miller, K.: Software testability: the new verification. IEEE Softw. **12**(3), 17–28 (1995). https://doi.org/10.1109/52.382180
32. Wanstrath, C., Hyett, P.J., Preston-Werner, T., Chacon, S.: Github, https://github.com/