BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Rough Sets Clustering and Supervised Learning Models with Applications in Regression Testing

PhD Thesis Summary

Scientific supervisor
Prof. Dr. Horia F. Pop

PhD Student
Arnold Szederjesi-Dragomir

2024

Keywords: Clustering, Rough Sets, Multi-Agent Systems, Supervised Learning, Test Case Prioritization, Continuous Integration

Contents

Chapter 1

Introduction

This Doctoral thesis, titled *Rough Sets Clustering and Supervised Learning Models with Applications in Regression Testing*, presents our research regarding the development of new machine learning models as well as their application in the software testing domain. Our contributions include the proposal of novel methods and approaches, primarily centered on the fields of rough clustering and test case prioritization. In rough clustering, we have developed new methodologies to enhance clustering techniques in uncertain environments, including the study of performance evaluation measures in such contexts and comprehensive performance evaluation strategies. For test case prioritization, this work presents several approaches that simplify and improve the efficiency of prioritization processes across diverse regression testing scenarios. Several of our approaches have been applied to real-world industrial datasets or even integrated in real-world industrial software projects.

1.1 Motivation

Machine learning [1], a powerful subset of artificial intelligence, transforms how we interact with and extract value from data. Computers learn to identify patterns and make predictions with minimal human guidance, revolutionizing industries across the board. From personalized experiences to medical breakthroughs, machine learning drives innovation in fields like finance, language processing, and many more. This technology is a pillar of modern progress – automating tasks, enhancing our understanding of data, and leading us towards a future of unprecedented efficiency and insight.

Unsupervised learning (or Clustering) [12], a crucial branch of machine learning, works with unlabeled data. Unlike its supervised counterpart, it does not rely on preclassified examples to guide its learning process, enabling algorithms to uncover hidden patterns, structures, and relationships within the data itself. This has the potential to allow them to perform a variety of tasks, such as clustering similar data points together, dimensionality reduction for data visualization, and anomaly detection. Unsupervised learning plays a vital role in diverse fields, from image and text analysis to market research and scientific discovery, offering a powerful tool for unlocking the hidden potential within data.

Software testing [4], the basis of software development, ensures the quality and reliability of the programs we rely on daily. It involves a meticulous process of evaluating software against its intended functionalities and identifying any potential errors or discrepancies. This vital step acts as a safety net, safeguarding users from encountering unexpected issues and ensuring software performs as expected.

As software evolves through new features and bug fixes, regression testing [21] becomes crucial. It involves re-running existing test cases, initially used to validate the original software build, to ensure that recent changes have not unintentionally introduced new issues. This continuous check-up is essential to maintain the stability and reliability of the software, especially as it undergoes modifications.

In the fast-paced world of software development, continuous integration (CI) [11] practices have emerged to streamline the development and testing cycle. CI involves regularly integrating code changes from various developers into a central repository, followed by automated testing to identify any potential issues early on. This continuous loop of integration and testing facilitates rapid feedback and quicker bug detection, ensuring software quality remains high throughout the development process.

Regression testing, while necessary, can be time-consuming due to the big volume of test cases involved. Test Case Prioritization (TCP) [7] emerges as a valuable method within the regression testing field. It involves re-ordering test cases based on various factors, such as their likelihood of uncovering faults, execution time, and historical data. This prioritization allows testers to focus on the most critical test cases first, optimizing the testing process, and minimizing the time spent identifying critical issues.

1.2 Objectives of the Thesis

The following objectives guided our research:

1. Study fuzzy and rough set theories in order to evaluate their effectiveness in uncertain contexts
2. Propose novel clustering models based on rough sets
3. Implement the rough clustering methods proposed
4. Study existing clustering evaluation methodologies
5. Propose and implement a methodology to evaluate rough clustering results
6. Evaluate the performance of these novel models using standard and novel methodologies
7. Apply the new rough clustering methods on standard and real-world datasets
8. Research the field of regression testing, especially test case prioritization (TCP)
9. Introduce a unified approach for TCP dataset building
10. Analyze state-of-the-art learning methods for TCP
11. Propose neural network-based models for TCP
12. Integrate the new neural network methods into a real-world solution and test on a real-world project
13. Apply the novel developed rough clustering approaches in software testing, more specifically test case prioritization

1.3 Original Contributions

Clustering [3], a central method in pattern recognition, faces challenges in real-world applications due to overlapping clusters, outliers, and complex data shapes. *This research introduces novel approaches to address these issues by incorporating concepts from rough set theory into various agglomerative clustering algorithms*.

Our novel approach, Agent Based Rough Clustering (ABARC) [6], a clustering algorithm based on rough sets, effectively identifies data points that potentially belong to multiple clusters (also called as rough instances), while simultaneously separating outliers from the core data. This process is driven by software agents operating in parallel, promoting computational efficiency. Additionally, we propose a methodology to evaluate rough clusters.

Moreover, we conducted experiments on standard datasets to demonstrate the significant role of choosing an appropriate similarity measure for achieving accurate clustering [15, 16], especially when dealing with overlapping data. Since standard datasets lack information about overlapping areas, we have implemented methods to extract this data for benchmarking purposes. This contribution highlights the potential of rough set theory and the selection of similarity measures for effective clustering in real-world scenarios.

Furthermore, we have also performed a comprehensive evaluation of our clustering methods using various different types of metrics [14]. The results show the promising performance of ABARC.

Regression testing in CI environments requires efficient execution of numerous test cases to ensure software quality. Test Case Prioritization (TCP) techniques address this challenge by reordering tests to prioritize those with a higher likelihood of uncovering faults, ultimately minimizing testing time and cost.

We investigate the effectiveness of a neural network-based model, NEUTRON [19], for TCP in CI. NEUTRON analyzes various features like execution duration, fault rate, and test history to intelligently prioritize test cases. The study demonstrates that NEUTRON outperforms random prioritization and achieves similar or better results compared to existing techniques, especially when considering larger test budgets (75% and 100%).

Building upon the success of NEUTRON, we also implemented MixTCP [17], an applied system designed to seamlessly integrate the model into software development workflows. Implemented in Elixir, MixTCP utilizes NEUTRON to prioritize tests, effectively improving fault detection and reducing testing time. Additionally, its modular architecture allows for future integration of alternative TCP solutions.

We also highlight the potential of NEUTRON and MixTCP for streamlining CI testing processes. NEUTRON tackles the challenges of test case prioritization in CI environments, while MixTCP offers developers a user-friendly and efficient solution to leverage this advanced model within their workflow. By optimizing regression testing, these advancements contribute to faster and more reliable software development cycles.

RoughTCP [5] is our rough clustering-based approach proposed for TCP, utilizing unsupervised learning through rough sets-based agglomerative clustering. This enables it to adapt to dynamic CI environments without requiring labeled data. Experiments demonstrate that RoughTCP outperforms most of the related work.

A novel framework [20] that we proposed for TCP considers various aspects such as traceabil-

ity information, context, and feature information. This framework aims to provide a more holistic view of the TCP problem. Experiments using a synthetic dataset demonstrate that the clustering method based on this framework consistently outperforms other methods, highlighting its effectiveness.

Overall, these advancements showcase the ongoing efforts to develop efficient and adaptable TCP techniques for optimized software development in CI environments.

To sum up, this thesis bridges the gap between theoretical advancements in machine learning and applications in software testing. Not only it offers theoretical contributions to the field of clustering but also demonstrates the value of these contributions by applying them to the real-world challenge of software testing.

The original contributions of the research, which are presented in Chapters 3, 4 and 5, are the following:

- **Fundamental research on Rough Clustering**
  - Novel clustering methods based on rough set theory - Agent Based Rough Clustering (ABARC) (Section 3.1) [6].
  - A new methodology on how to evaluate results in uncertain contexts (Section 3.1) [6].
  - Comparison study of multiple similarity metrics, especially in rough environments using ABARC and methodology to evaluate them (Section 3.2) [15, 16].
  - Analysis of performance evaluation metrics (internal, external and rough) and related experiments on ABARC in unknown conditions (Section 3.3) [14].
  - Experiments on ABARC and on the similarity metrics on various standard datasets (Sections 3.1, 3.2) [6, 15, 16].
- **Fundamental research on Test Case Prioritization**
  - Novel neural network-based models - Neural Network-based Test Case Prioritization in Continuous Integration (NEUTRON) (Section 4.1) [19].
  - A unified, unique and comprehensive approach to modern Test Case Prioritization that considers all the principles used throughout multiple regression testing use cases (Section 5.2) [20].
  - A unified synthetic dataset and the demonstration of its effectiveness using ABARC in the context of Test Case Prioritization (Section 5.2) [20].
  - Rough Clustering based unsupervised learning approaches for Test Case Prioritization - RoughTCP (Section 5.1) [5].
- **Applicative research**
  - Application of NEUTRON on multiple industrial datasets (Section 4.1) [19].
  - Integration of NEUTRON in a real-world solution - MixTCP (Section 4.2) [17].
  - Application of NEUTRON through MixTCP on a real-world industrial project (Section 4.2) [17].
  - Experiments on RoughTCP using standard industrial datasets (Section 5.1) [5].

1.4 List of Publications

The ranking of publications was performed according to the recent journal and conference lists published by CNATDCU (National Council for the Recognition of University Degrees, Diplomas and Certificates) which are applicable since 1st of October 2018.

1. [6] R. D. Găceanu, **A. Szederjesi-Dragomir**, H. F. Pop, and C. Sârbu, "ABARC: An Agent-based Rough Sets Clustering Algorithm" Intelligent Systems with Applications, vol. 16, p. 200117, 2022.
   Indexed Web of Science and Scopus, Rank C, 1 point.

2. [15] **A. Szederjesi-Dragomir**, R. D. Găceanu, H. F. Pop, and C. Sârbu, "A Comparison Study of Similarity Measures in Rough Sets Clustering" in Proceedings of the 2019 IEEE 15th International Scientific Conference on Informatics (INFORMATICS 2019). Editors: William Steingartner, Štefan Korečko, Anikó Szakál, pp. 399 – 405, IEEE Press, Poprad, Slovakia, Nov 20 – 22, 2019.
   Indexed IEEE, Rank D, 0.5 points.

3. [16] **A. Szederjesi-Dragomir**, R. D. Găceanu, H. F. Pop, and C. Sârbu, "Experiments on Rough Sets Clustering with Various Similarity Measures" IPSI BGD TRANSACTIONS ON INTERNET RESEARCH, vol. 16, pp. 75–83, 2020.
   Indexed Web of Science, Rank D, 0 points.

4. [19] A. Vescan, R. D. Găceanu, and **A. Szederjesi-Dragomir**, "Neural Network-Based Test Case Prioritization in Continuous Integration" in 38th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW), pp. 68–77, IEEE, IEEE Computer Society, sep 2023.
   Indexed IEEE, Workshop at Rank A conference, 6 points.*

5. [14] **A. Szederjesi-Dragomir**, "A Comprehensive Evaluation of Rough Sets Clustering in Uncertainty Driven Contexts". Studia Universitatis Babes-Bolyai Informatica 69, 1 (2024), 41–56.
   Indexed Mathematical Reviews, Rank D, 1 point.

6. [20] A. Vescan, R. D. Găceanu, **A. Szederjesi-Dragomir**, "Embracing Unification: a Comprehensive Approach to Modern Test Case Prioritization" in Proceedings of the 19th International Conference on Evaluation of Novel Approaches to Software Engineering - ENASE 2024, INSTICC, Angers, France, 29 april 2024, SciTePress, pp. 396–405.
   Indexed IEEE, short paper at Rank B conference, 2.66 points.

7. [17] **A. Szederjesi-Dragomir**, R. D. Găceanu, and A. Vescan, "Industrial Validation of a Neural Network Model Using the Novel MixTCP Tool" in Proceedings of the 19th International Conference on Evaluation of Novel Approaches to Software Engineering - ENASE 2024, INSTICC, Angers, France, 29 april 2024, SciTePress, pp. 110–119.
   Indexed IEEE, Rank B conference, 4 points.

Total points from publications: 15.16 points.

Submitted Papers

1. [5] R. D. Găceanu, **A. Szederjesi-Dragomir**, and A. Vescan, "Leveraging Rough Sets for Enhanced Test Case Prioritization in a Continuous Integration Context" in 7th Workshop on Validation, Analysis and Evolution of Software Tests (VST 2024), at the 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER 2024), IEEE, Rovaniemi, Finland, 12 march 2024, **accepted for publication**.
   Indexed IEEE, Workshop at Rank A conference, 4 points.

Chapter 2

Theoretical Background

In this Chapter we present the background and state of the art contributions that we have leveraged in our research. It gives more context on functional and concurrent programming, multi-agent systems, unsupervised learning, rough sets and finally about automated software testing, especially test case prioritization. Next we will describe the structure of this chapter in the PhD thesis divided into sections.

Section 1, named *Functional concurrent programming*, presents how functional and concurrent programming works, introduces the programming languages and the web framework we used, gives us ideas how concurrency in those languages work and explains why we have chosen these technologies.

Section 2, named *Multi-agent systems*, describes multi-agent systems, starts with the concept of an agent, continues with agent types and finally it shows how agents can work together to achieve different goals by listing concrete applications.

Section 3, named *Supervised Learning*, defines supervised learning, enumerates some of its methods, and finally describes neural networks in more detail.

Section 4, named *Unsupervised Learning*, defines unsupervised learning (clustering), outlines its types, specifies what hierarchical clustering is and finally introduces rough sets and explains how they can be used when applying hierarchical clustering.

Section 5, named *Rough Sets*, presents the rough set theory formally, followed by applications of it, especially in the machine learning domain, and finally the state of the art.

Section 6, named *Automated Software Testing*, is about automated software testing. It presents one of the most popular testing method: regression testing. Then it describes the problem of test case prioritization.

Chapter 3

Novel Approaches to Rough Sets Clustering

This chapter presents our theoretical contributions to Clustering using Rough Sets. These contributions include an original clustering algorithm based on multi-agent systems and rough sets and experiments on similarity measures for this new approach.

Section 3.1, introduces a novel clustering algorithm that uses rough sets and software agents to enhance both the accuracy on vague datasets and the time needed in order for the agglomerative method to converge. This section also proposes a new methodology to evaluate clustering results that have uncertain (rough) or outlier instances, coined as hybrid instances.

Section 3.2 analyses the impact of various similarity distance metrics and the methodology used to evaluate their performance both based on well-known measures, like accuracy and on a novel way, like evaluation based on rough instances.

Section 3.3 seeks to conduct an in-depth comparison of the ABARC algorithm with a range of both supervised and unsupervised learning algorithms, using a variety of performance metrics to evaluate the effectiveness of each algorithm on commonly used datasets.

Section 3.4 concludes the chapter and presents options for future work.

The approaches introduced in this chapter represent original research contributions published in [6, 15, 16, 14].

3.1 Agent Based Rough Clustering

The content of this section is derived from the original paper [6].

Clustering is an important task in pattern recognition with many applications in natural sciences and healthcare. However, in practical scenarios, it is often the case that the data cannot be easily separated into well distinguished groups for several reasons like: the shape of clusters, the presence of outliers, or the overlapping clusters problem (instances that may belong to more than one cluster). In order to handle such issues, we propose an agglomerative clustering approach which identifies instances that may belong to more than one cluster and clearly separates the outliers from the rest of the instances by integrating concepts from rough set theory. The whole grouping and regrouping process is driven by software agents executing in parallel. Our approach is computational friendly and experiments on standard data sets indicate its advantages.

This section presents our approach to clustering. We introduce **ABARC** (**A**gent **BA**sed **R**ough **C**lustering), algorithms which address the overlapping clusters problem by modeling clusters using notions from the rough set theory. They successfully identify outliers and, by using software agents, it is also scalable. Besides this clustering approach, we also introduce what we believe to be an objective methodology for evaluating the quality of a clustering result in case of overlapping clusters and outliers.

The main clustering procedure is described in Algorithm 3.1, where X is the data set containing the instances to be clustered, i_{max} and λ are integers denoting the number of trials for specific tasks (details bellow), σ_1 is the similarity limit and δ is a distance metric (for example, the Euclidean distance). The similarity limit, σ_1, is specific to every data set and it is a real number denoting the maximum value up to which two instances are considered similar (they surely belong to the same group). The first step is to initialize the agents and associate one agent to each instance. Also, each agent is assigned to a different cluster, so, at the beginning, the number of clusters is equal to the number of agents which is equal to the number of instances. Agents are executed in parallel, in separate processes, and this behavior is indicated by the $\|$ operator from line 4.

Algorithm 3.1: Agent Clustering

```
   Data: X, i_max, λ, σ_1, δ
   Result: RC //the set of rough clusters
1  @ Let AG be the set of agents
2  for i = 1, i_max do
3  |  for k = 1, |AG| do
4  |  |  || doCluster(agent_k, λ, σ_1, δ, AG)
5  |  end
6  end
```

Algorithm 3.2 shows the asynchronous behavior of each agent: given an $agent_k$, it tries to find similar fellows with respect to σ_1 and δ by direct message exchange. One it finds a similar agent, it moves to its cluster (line 3).

Algorithm 3.2: Do Cluster

```
   Data: agent_k, λ, σ_1, δ, AG
1  sa_k = searchForSimilar(agent_k, λ, σ_1, δ, AG)
2  if sa_k ≠ null then
3  |  changeCluster(agent_k, sa_k)
4  end
```

Algorithm 3.3 describes the procedure of finding a similar agent. The argument λ denotes the maximum number of attempts the agent should perform in order to find a similar one. On line 4, an agent is selected in a non-deterministic way as indicated by the $\bigcap$ operator and, if the two agents are not in the same cluster, their $similarity$ is computed. If this value is bellow the similarity limit, σ_1, then a similar agent is found and the function terminates by returning the $selectedAgent$. Otherwise the search continues and after λ failed attempts of finding similar agents (which are not located in the current cluster) the function returns $null$ meaning that either there are no agents similar to the given one ($agent_k$) or the process simply took too much time in which case the task is left to other agents or to another iteration (line 2 from Algorithm 3.1) when the search process is probably faster since there are less clusters. The $computeSimilarity$ function from line 6 computes the similarity between two agents given a distance metric δ. If this value is bellow the similarity limit, σ_1, it means that the two agents $certainly$ belong to the same cluster so they are in the $lower\ approximation$ of the current cluster.

Algorithm 3.4 represents the second phase of our approach. Since agents are grouped together only if they should certainly belong to the same cluster (based on the similarity limit, σ_1), the

Algorithm 3.3: Search for Similar

Data: $agent_k, \lambda, \sigma_1, \delta, \mathcal{AG}$
Result: sa //similar agent
1 **if** $\lambda = 0$ **then**
2 | **return** $null$
3 **end**
4 $\bigcap\{selectedAgent = a_j, \forall j = \overline{1, |\mathcal{A}|}, a_j \in \mathcal{AG}\}$
5 **if** $getCluster(agent) \neq getCluster(selectedAgent)$ **then**
6 | $similarity = computeSimilarity(agent, selectedAgent, \delta)$
7 | **if** $similarity \leq \sigma_1$ **then**
8 | | **return** $selectedAgent$
9 | **else**
10 | | **return** $searchForSimilar(agent, \lambda - 1, \sigma_1, \delta, \mathcal{AG})$
11 | **end**
12 **else**
13 | **return** $searchForSimilar(agent, \lambda - 1, \sigma_1, \delta, \mathcal{AG})$
14 **end**

first phase of our approach (Algorithms 3.1, 3.2 and 3.3) will probably produce a large number of clusters. The second phase (Algorithm 3.4) unifies similar clusters producing rough clusters. The algorithm receives as a first argument a set of cluster representatives. A *cluster representative*, $\mathcal{R}_k = \langle C_k, \mathcal{A}(C_k) \rangle$, is a tuple where $\mathcal{A}(C_k)$ is the centroid of the cluster C_k and it is computed as follows:

$$\mathcal{A}(C_k) = \frac{1}{|C_k|} \sum_{x^i \in C_k} x^i \tag{3.1}$$

The second parameter, σ_2, is a *rough* similarity limit denoting up to what point two agents are *possibly* similar. This parameter is different from the σ_1 value (used in Algorithms 3.1, 3.2 and 3.3) which denotes the point up to which two agents are *surely* similar. The last argument, $unified$, denotes the set of unified clusters and it is initially equal to the empty set. Cluster similarity is computed based on the centroid values in the same fashion agent similarity is performed. If a representative is similar to several ones then the corresponding data will belong to several clusters in the upper approximation. The result is a set of representatives of the unified clusters. The $updateRepresentatives$ function from line 8 recalculates the representatives using Equation 3.1.

Even after executing Algorithm 3.4 there might be a significant number of clusters remaining, but most of them are normally composed of a very small number of entities which are not similar to either of the "normal" clusters. The instances from these small clusters will be marked as possible *outliers*. Nevertheless, in Algorithm 3.5, the third phase of our approach, we will assign them to the closest cluster and we get the final clustering structure.

Algorithm 3.5 receives as input data the $clusters$, as resulted after applying Algorithm 3.4. In line 1, the outliers (which are themselves clusters) are separated from the 'normal' clusters. This decision is based on the value of ε and the cluster size: if the number of instances from a cluster is less then ε then the cluster is marked as an outlier. The value of ε is set to 5% from the total number of instances in the data set. In line 2, each outlier is unified with the closest 'normal' cluster, based on the cluster representatives. A more comprehensive explanation of the algorithms, alongside with our novel methodology to evaluate rough clusters can be found in the thesis.

In order to evaluate the quality of our approach, we use several cluster evaluation measures

Algorithm 3.4: SimilarClusterUnification

Data: $representatives$, σ_2, $unified$
Result: $unified$
if $representatives = \emptyset$ **then**
  return $unified$
end
$\mathcal{R}_k = first(representatives)$
$\mathcal{S} = getSimilar(\mathcal{R}_k, representatives \setminus \{\mathcal{R}_k\}, \sigma_2)$
if $\mathcal{S} \neq \emptyset$ **then**
  $updateCluster(\mathcal{R}_k\langle C_k\rangle, \mathcal{S})$
  $newRepresentatives = updateRepresentatives(representatives)$
  return $SimilarClusterUnification(newRepresentatives, \sigma_2, \emptyset)$
else
  return $SimilarClusterUnification(representatives \setminus \{\mathcal{R}_k\}, \sigma_2, unified \cup \{R_k\})$
end

Algorithm 3.5: Outlier Elimination

Data: $clusters$, ε
Result: $finalClusters$
$\{outliers, clusters\} = detectOutliers(clusters, \varepsilon)$
$finalClusters = joinOutliers(outliers, clusters)$
return $finalClusters$

presented in the thesis. For each data set, we execute the algorithm 30 times and we present the average value for each evaluation measure in Table 3.1. We would like to mention that the number of executions was chosen to be 30 only because we would like to be consistent with other approaches we compare with. In line $Official$ from Table 3.1 we show the index values for the clusters given in the official data set documentation. In line $ABARC$ we show the average index values for the clusters obtained by our algorithm with all hybrid data included. Line $ABARC-O$ shows the average index values for our clusters after eliminating the outliers.

Table 3.1: Cluster quality measures.

		$DB\downarrow$	$DN\uparrow$	$SI\uparrow$	$Entropy\downarrow$	$ARI\uparrow$	$Accuracy(\%)\uparrow$
Iris	$Official$	0.511	2.484	0.624	**0**	**1**	**100**
	$ABARC$	0.464	3.012	0.641	0.116	0.77	95.556
	$ABARC-O$	**0.354**	**3.616**	**0.716**	0.067	0.92	97.56
Seeds	$Official$	0.527	2.699	0.561	**0**	**1**	**100**
	$ABARC$	0.463	3.447	0.614	0.293	0.51	90.952
	$ABARC-O$	**0.21**	**5.972**	**0.772**	0.073	0.6	98.121
Wine	$Official$	0.98	1.474	0.465	**0**	**1**	**100**
	$ABARC$	0.968	1.6	0.48	0.11	0.65	96.985
	$ABARC-O$	**0.425**	**3.804**	**0.677**	**0**	0.85	**100**

As it may be seen in Table 3.1, the cluster structure proposed in the official documentation outperforms the results proposed by us only in terms of accuracy, ARI and entropy (of course). The best results for each index is marked in bold face. Even without eliminating the outliers, all results (except for the accuracy, ARI and entropy) are better compared to the official ones, but after eliminating the outliers the results are significantly improved in some cases. More experiments and evaluations can be found in the thesis.

3.2 Similarity Measures for Rough Sets Clustering

The content of this section is derived from the original papers [15, 16].

The proper selection of the similarity measure may be of paramount importance for the clustering result especially if the clusters overlap. Therefore, the aim of this section is to study the influence of some widely known similarity measures on the clustering structure. The analysis uses a clustering algorithm based on rough sets which is able to discover hybrid data (outliers and instances that are close enough to more than one cluster). In this section, we also examine the similarity measures influence on the overlapping regions as well. Since the standard datasets do not offer specific information regarding the overlapping areas, we also propose an approach for extracting this data in order to be used for benchmarking purposes. Experiments conducted on standard data sets outline the importance of the proper similarity measure selection.

In order to evaluate the influence of a certain similarity measure on the rough instances, we compare the rough instances reported by the algorithm described in Section 3.1.

Given a similarity measure, we execute the algorithm several times (N) on a certain data set and we collect the reported rough instances from each execution.

For each rough instance x^i we compute the *occurrence rate*: $occ_{x^i} = \frac{n}{N}$ where n denotes the number of occurrences of the instance in the series of N executions.

For a given distance d, we compute the *rough score* as shown in Definition 1.

Definition 1 *The* **rough score** *of a given similarity measure is the ratio between the sum of the* occurrence rates *of the validated rough instances and the sum of all* occurrence rates*:*

$$\mathcal{R}_{sc} = \frac{\sum_{i=1}^{|RI_d|} \mathbb{1}_{RI}(x^i) \cdot occ_{x^i}}{\sum_{i=1}^{|RI_d|} occ_{x^i}},$$

where:

- RI_d *is the set of rough instances reported by the clustering algorithm for a certain similarity measure (or distance,* d*)*
- RI *is the set of actual rough instances produced by the algorithm from Section 3.1*
- $\mathbb{1}_{RI}$ *is the indicator function of* RI*:*

$$\mathbb{1}_{RI}(x^i) = \begin{cases} 1, & \text{if } x^i \in RI \\ 0, & \text{otherwise.} \end{cases}$$

As seen in Definition 1, the *rough-score* depends on a fixed parameter κ denoting the confidence of each rough instance x^i — so only the instances having an *occurrence rate* higher than κ are taken into account. Given a similarity measure, we compute the rough score for several values of κ. The rest of our methodology can be found in the PhD thesis.

Talking about results, the rough scores can be found in Table 3.2. It seems that the best results (high rough scores for high confidence values) are obtained in most cases for values of p close to 2.3. For example, for Wine and Seeds the best results are obtained for $p = \sqrt{2}$, while for the other datasets $p = 2\sqrt{2}$ gives the best results, but these values of p are both close to 2.3. Other results can be found in the thesis.

Table 3.2: Rough scores with minimum confidences (κ) ranging between 0 and 1 with a step of 0.1.

	Similarity measure	***0***	***0.1***	***0.2***	***0.3***	***0.4***	***0.5***	***0.6***	***0.7***	***0.8***	***0.9***	***1***
Iris	Manhattan	2.61	11.0	25.5	25.5	25.5	20.0	20.0	NaN	NaN	NaN	NaN
	Chebyshev	6.36	9.5	13.13	14.31	19.5	11.0	11.0	0.0	0.0	0.0	0.0
	Euclidean	5.9	38.5	38.5	38.5	38.5	51.33	51.33	45.0	45.0	45.0	0.0
	Squared Euclidean	14.88	15.25	18.39	18.39	59.33	89.0	89.0	89.0	89.0	98.0	NaN
	Minkowski $p = \sqrt{2}$	5.56	36.0	36.0	36.0	36.0	48.0	46.0	46.0	46.0	46.0	0.0
	Minkowski $p = 2.3$	15.71	18.17	27.25	43.6	43.6	54.5	80.0	80.0	86.0	NaN	NaN
	Minkowski $p = 2\sqrt{2}$	7.61	18.88	25.17	31.71	62.0	62.0	62.0	62.0	93.0	98.0	NaN
	Minkowski $p = 3$	6.12	16.0	17.33	29.71	59.33	59.33	59.33	89.0	89.0	96.0	NaN
Wine	Manhattan	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	Chebyshev	7.73	11.94	20.24	20.0	20.29	33.33	33.33	33.33	50.0	100.0	100.0
	Euclidean	5.36	8.0	10.47	15.33	23.0	0.0	0.0	0.0	0.0	NaN	NaN
	Squared Euclidean	3.65	6.83	10.91	6.4	0.0	0.0	0.0	NaN	NaN	NaN	NaN
	Minkowski $p = \sqrt{2}$	5.72	8.66	10.63	11.47	14.0	16.55	10.57	12.33	0.0	NaN	NaN
	Minkowski $p = 2.3$	4.99	8.87	9.39	10.67	0.0	0.0	0.0	0.0	0.0	NaN	NaN
	Minkowski $p = 2\sqrt{2}$	4.03	6.33	8.29	8.5	0.0	0.0	0.0	0.0	NaN	NaN	NaN
	Minkowski $p = 3$	5.15	7.04	10.74	23.33	16.0	0.0	0.0	0.0	0.0	NaN	NaN
Seeds	Manhattan	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	Chebyshev	1.74	2.51	7.4	4.75	0.0	0.0	0.0	0.0	NaN	NaN	NaN
	Euclidean	3.62	4.87	10.34	17.5	18.62	19.0	15.0	NaN	NaN	NaN	NaN
	Squared Euclidean	4.49	6.94	8.4	8.64	27.56	41.6	52.0	86.0	86.0	NaN	NaN
	Minkowski $p = \sqrt{2}$	4.97	7.8	9.35	13.52	17.45	14.86	22.29	52.0	40.0	0.0	0.0
	Minkowski $p = 2.3$	3.58	6.07	11.64	21.62	25.08	37.6	33.5	35.0	NaN	NaN	NaN
	Minkowski $p = 2\sqrt{2}$	4.09	7.01	11.74	20.46	24.11	29.56	41.0	41.0	31.33	31.33	0.0
	Minkowski $p = 3$	3.41	4.73	7.04	8.38	18.92	7.78	14.0	14.0	0.0	0.0	NaN
Ionosphere	Manhattan	2.0	1.27	0.0	0.0	0.0	0.0	0.0	0.0	0.0	NaN	NaN
	Chebyshev	14.5	19.33	19.33	19.33	29.0	29.0	NaN	NaN	NaN	NaN	NaN
	Euclidean	2.65	6.45	0.0	0.0	0.0	0.0	0.0	0.0	0.0	NaN	NaN
	Squared Euclidean	1.85	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	NaN	NaN
	Minkowski $p = \sqrt{2}$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	NaN	NaN	NaN
	Minkowski $p = 2.3$	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	NaN	NaN
	Minkowski $p = 2\sqrt{2}$	2.29	37.0	37.0	37.0	74.0	74.0	74.0	74.0	NaN	NaN	NaN
	Minkowski $p = 3$	5.0	8.24	0.0	0.0	0.0	0.0	0.0	0.0	NaN	NaN	NaN
Ecoli	Manhattan	3.05	7.04	6.62	0.0	0.0	0.0	0.0	0.0	0.0	0.0	NaN
	Chebyshev	4.64	22.93	47.0	52.0	56.0	56.0	56.0	48.0	48.0	48.0	NaN
	Euclidean	2.88	8.98	11.46	13.45	11.14	15.6	26.0	78.0	NaN	NaN	NaN
	Squared Euclidean	3.96	10.48	14.16	18.22	27.2	56.0	NaN	NaN	NaN	NaN	NaN
	Minkowski $p = \sqrt{2}$	3.23	8.57	13.83	28.4	47.33	47.33	47.33	37.0	NaN	NaN	NaN
	Minkowski $p = 2.3$	4.61	12.11	16.35	17.87	19.0	22.4	0.0	NaN	NaN	NaN	NaN
	Minkowski $p = 2\sqrt{2}$	3.31	9.31	13.43	19.5	26.0	39.0	78.0	78.0	NaN	NaN	NaN
	Minkowski $p = 3$	2.23	9.5	10.36	10.29	24.0	24.0	24.0	36.0	NaN	NaN	NaN
Glass	Manhattan	10.0	34.0	34.0	34.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	Chebyshev	8.0	16.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	Euclidean	6.44	34.0	34.0	34.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	Squared Euclidean	9.2	34.0	34.0	34.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	Minkowski $p = \sqrt{2}$	8.53	25.5	25.5	36.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	Minkowski $p = 2.3$	11.67	17.0	34.0	34.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	Minkowski $p = 2\sqrt{2}$	50.0	50.0	50.0	80.0	80.0	80.0	80.0	80.0	80.0	NaN	NaN
	Minkowski $p = 3$	35.0	35.0	52.0	52.0	72.0	72.0	72.0	72.0	NaN	NaN	NaN

3.3 Uncertainty Driven Clustering Evaluation

The content of this section is derived from the original paper [14].

This section aims to perform a comprehensive comparison of the ABARC algorithm against several supervised and unsupervised learning algorithms, employing a suite of performance metrics to assess each algorithm's efficacy across standard datasets. Through this comparative analysis, our aim is to show the strengths and limitations of the ABARC algorithm and its counterparts, thereby contributing to the ongoing research on optimal clustering approaches in the context of data uncertainty.

This comprehensive analysis includes three types of metrics: external, internal and rough evaluation metrics. The external ones we considered are: Accuracy, Precision, Recall, F1-Score, Macro F1-Score, Weighted Average F1-Score, Micro F1-Score and Kappa Score. Comparisons using these metrics were conducted on the Iris, Seeds and Wine datasets. The internal ones used are: Purity, Entropy, V-Measure. For the comparisons we again used the same three datasets. In the final we have the following rough metrics: Average Accuracy (α index), Average Roughness (ρ index), Accuracy of Approximation (α^* index), and Quality of Approximation (γ index). These evaluation metrics were used only on the Iris and Wine datasets, because we did not have comparison subjects for the Seeds dataset.

Considering the results, in Table 3.3 we can observe that on Iris the accuracy and purity drops a bit as we eliminate outliers and rough, but the entropy and the V-Measure after dropping both of them is significantly better, which makes us assume that outliers and rough instances do not really affect homogeneity but they affect completeness. On the seeds dataset we cannot see any real difference when eliminating them, thus they do not affect our performance. Finally, on wine we can see all metrics improve, entropy and V-Measure improves significantly, so on this dataset eliminating them makes our results almost perfect regardless of the metric used. The rest of the performance measurements can be found in the thesis.

Table 3.3: Unsupervised performance measurements for the Iris, Seeds and Wine datasets.

	Case Study	Inst	Acc	Entropy	Purity	V
Iris	Clusters with hybrids	150	98.66%	0.0803	0.987	0.733
	Clusters without outliers	139	98.56%	0.0847	0.986	0.717
	Clusters without outliers and rough	126	98.41%	0.0204	0.984	0.932
Seeds	Clusters with hybrids	210	92.857%	0.0839	0.929	0.721
	Clusters without outliers	190	92.105%	0.0863	0.921	0.711
	Clusters without outliers and rough	178	91.573%	0.0829	0.916	0.719
Wine	Clusters with hybrids	178	97.753%	0.0569	0.978	0.8
	Clusters without outliers	157	99.363%	0.0418	0.994	0.854
	Clusters without outliers and rough	148	99.324%	0.0088	0.993	0.97

3.4 Conclusions and Further Work

In this chapter, we introduced an agglomerative clustering algorithm that utilizes rough set theory to identify hybrid data, such as outliers or rough instances. Hybrid data includes instances with traits of multiple clusters, providing deeper insights than other methods. Identifying outliers helps analysts remove potential measurement errors, improving classification accuracy. Rough in-

stances can indicate new classes or mutant individuals in datasets like Iris. Our scalable algorithm runs quickly on standard computers with large datasets.

We also presented an objective methodology to assess hybrid data quality, outperforming many traditional cluster quality measures. Our validation methodology highlights our algorithm's performance against documented dataset information. Additionally, we examined the impact of various similarity measures on clustering accuracy and rough instances, proposing a method to determine rough instances for benchmarking.

Our algorithm outperforms other classification techniques in Accuracy, Entropy, and Adjusted Rand Index, especially after outlier removal. The results demonstrate the algorithm's robust performance and its unique ability to identify hybrid data. We extensively evaluated the Agent BAsed Rough sets Clustering (ABARC) algorithm using standard datasets and compared it against multiple methods, analyzing the effects of different metrics in unpredictable contexts.

Experiments show the Minkowsky distance with $p = 2.3$ is optimal for accuracy and rough instances, likely due to similar spherical cluster shapes in all datasets. Removing hybrids enhances ABARC's performance, particularly in Iris and Wine datasets, underscoring the importance of hybrid data detection in uncertain environments. The findings also stress the need for suitable validation metrics for clustering in vague or unpredictable scenarios.

Future work includes experimenting with datasets featuring larger overlapping regions and varied cluster shapes, and focusing on outlier management by validating results and examining the influence of similarity measure selection.

Chapter 4

Contributions to Test Case Prioritization in Continuous Integration Context

This chapter presents our research on test case prioritization, and our contributions using a neural network-based approach on industrial datasets and integrating it in an applied solution for a real-world project.

Section 4.1 describes our approach to test case prioritization using neural networks (NEUTRON), as well as its application and performance on industrial datasets, like Google Shared Dataset of Test Suite Results.

Section 4.2 presents the application and evaluation of the NEUTRON approach in a real-world industrial project.

Section 4.3 concludes the chapter and states our potential future work.

The approaches introduced in this chapter represent original research contributions published in [19, 17].

4.1 Neural Network Based Test Case Prioritization in Continuous Integration

The content of this section is derived from the original paper [19].

In continuous integration environments, the execution of test cases is performed for every newly added feature or when a bug fix occurs. Therefore, regression testing is performed considering various testing strategies. The Test Case Prioritization (TCP) approach considers reordering test cases so that faults are found earlier with a minimum execution cost.

The purpose of the section is to investigate the impact of neural network-based classification models to assist in the prioritization of test cases. Three different models are employed with various features (duration, fault rate, cycles count, total runs count) and considering information at every 30 cycles or at every 100 cycles.

The results obtained emphasize that the NEUTRON approach finds a better prioritization with respect to NAPFD (normalized average percent of the detected fault) than random permutation and is comparable with the solutions that used either duration or faults, considering that it combines both values. Compared to other existing approaches, NEUTRON obtains similar competitive results when considering a budget of 50% and the best results when considering budgets of 75% and 100%.

After presenting the state of the art in test case prioritization, we present our different approaches to it, followed by the description of the different neural network models (seen on Figure 4.1) we use and the design of experiments (seen on Figure 4.2). We have considered three indus-

trial datasets: two from ABB Robotics Norway [1] (Paint Control and IOF/ROL, for testing complex industrial robots) and Google Shared Dataset of Test Suite Results (GSDTSR) [2]. As for the metrics for the analysis we NAPFD, which can adapt better to use cases where not all test faults are found. Labelling of the training data is done through an inteligent agent, that can simulate the experience and knowledge of an expert in this domain. A more detailed description and explanation of our approaches can be found in the thesis.

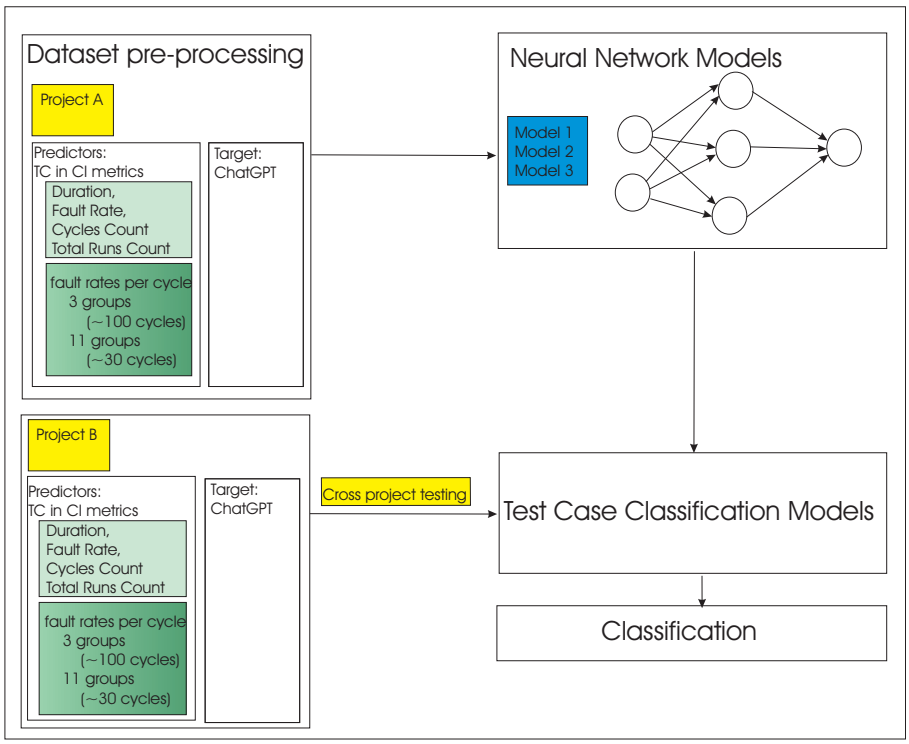

Figure 4.1: Overview of the neural networks-based models for TCP in Continous Integration

For all experiments, the training was carried out in the *IOF/ROL* project (since it was the most balanced dataset among the available ones) and testing was carried out in the *Paint Control* and *GSDTSR* projects.

Experiment 1

The current experiment considers for each test case the four general features mentioned above, together with the fault information for each cycle. The neural network uses the provided data for every 30 cycles.

As can be observed in Figure 4.3, the best results, in the case of 50% budget, are obtained by RETECTS [13] and COLEMAN [8]. It should be noted that NEUTRON obtains better results than the *Random* solutions and also better than *Sorted by Duration* and *Sorted by Fault Rate* in each of the tested projects, however, the NEUTRON solution embedded both features regarding duration and fault rate.

In Figure 4.4, it is shown that NEUTRON obtains the best solution for 75% budget in the case of testing *Paint Control* and for both testing projects in the case of the 100% budget. Table 4.1 contains the values of the NAPFD obtained for the previous papers Elbaum's approach [2], RETECTS [13],

[1] https://new.abb.com/products/robotics
[2] https://bitbucket.org/HelgeS/atcs-data/src

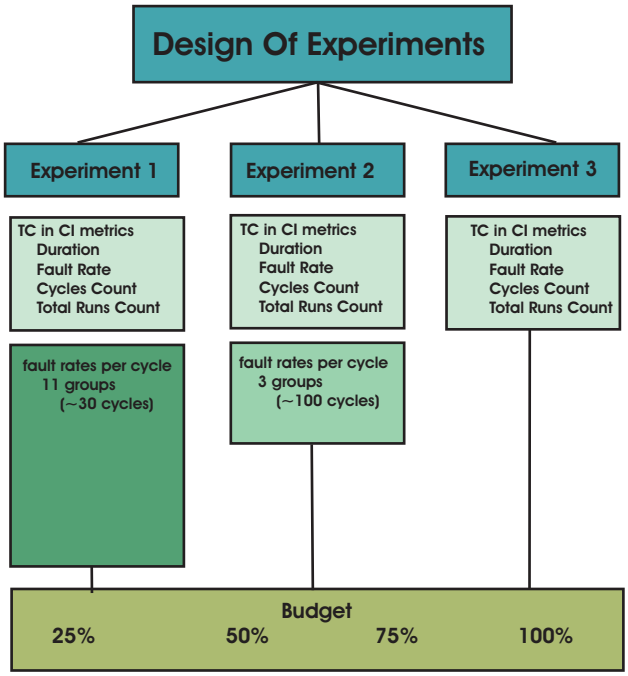

Figure 4.2: Design of experiments

COLEMAN [8], LeanRnTeC [9] and the results for *NEUTRON* approach, along with the results for the initial permutation, random and soft by duration or faults. It should be stated that the complete values exist only for the 50% budget. For the 25% and 75%, we have considered in the table the results of previous solutions for 10% and 80% respectively. The rest of the experiments can be found in the thesis.

Table 4.1: Experiment 1

Project	NAPFD							
	Initial	Random	Sort Fault Rate	Sort Duration	NEUTRON	Elbaum	RETECS	COLEMAN
25%-Paint Control	0.275280	0.320224	0.067415	0.567415	0.251276		0.915	0.915
25%-GSDTSR	0.160602	0.208414	0.798651	0.692944	0.486858		0.9911	0.9893
50%-Paint Control	0.584485	0.589887	0.372999	0.780898	0.600487	0.9145	0.915	0.915
50%-GSDTSR	0.409727	0.486987	0.999755	0.882094	0.917503	0.9891	0.9911	0.9893
75%-Paint Control	0.666328	0.864736	0.700434	0.949570	0.937796		0.9162	0.9171
75%-GSDTSR	0.672442	0.697674	0.999878	0.945154	0.987035		0.9921	0.9893
100%-Paint Control	0.988700	0.980494	1	0.972667	0.994003			
100%-GSDTSR	0.982327	0.974485	0.999904	0.959203	0.998308			

4.2 MixTCP: an Approach for Enhancing Software Development Experience

The content of this section is derived from the original paper [17].

Test Case Prioritization (TCP) is crucial in the fast-paced world of software development to speed up and optimize testing procedures, particularly in Continuous Integration (CI) setups. This section aims to first validate a state-of-the-art neural network model to TCP in CI environments, by applying it into a real-world industrial context, and second to propose MixTCP, an applied solution that integrates the neural network model and significantly enhances the regression testing experience from the software developer perspective. It is implemented in the Elixir programming language and employs the NEUTRON model, a state-of-the-art approach that uses

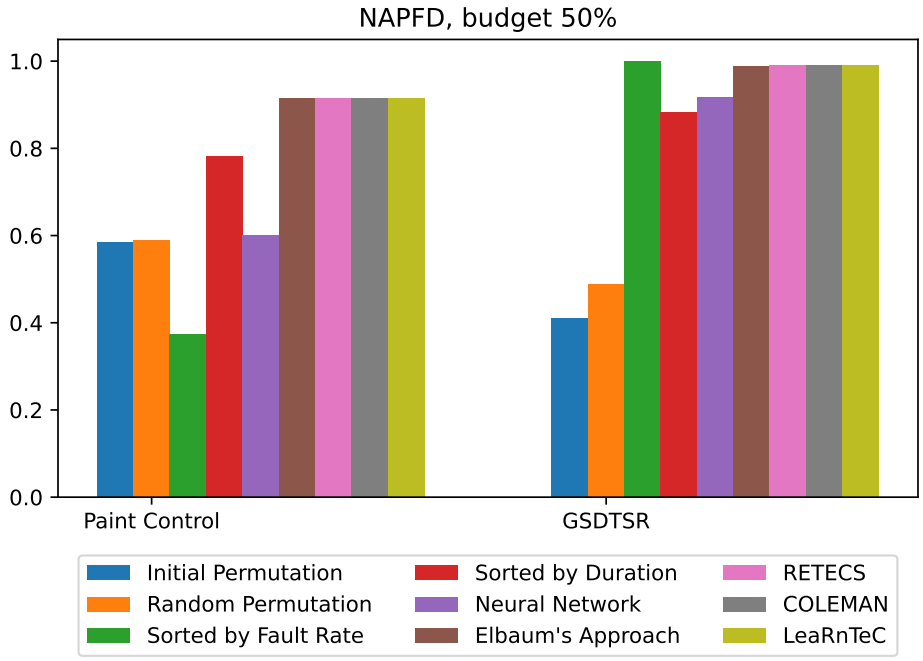

Figure 4.3: Experiment 1 with approximately 11x30 cycles, considering 50% budget

neural networks to intelligently prioritize test cases, effectively improving fault detection and reducing testing time. The system is composed of loosely coupled components (Mix TCP task, TCP Server, and NEUTRON model as seen on Figure 4.5), thus enabling the integration of other Test Case Prioritization solutions too. The results show that MixTCP has the potential to be a valuable asset to modern software development methods, offering software engineers a more efficient, a more user-friendly, and an overall easier to integrate TCP approach.

We firstly present our motivation and the state of the art, and then we introduce our MixTCP approach (as seen also on Figure 4.5). We applied TCP and NEUTRON (as described in Section 4.1) on a real-world Mix project [22]. In the following parts we detail our architecture, specify how to use this software, and discuss the advantages of it, which shortly are: integration with NEUTRON (a state-of-the-art approach for TCP), scalability, empirical validation in industrial settings and user experience and flexible integration. All of this can be seen in more detail in the thesis.

We have designed an experiment that considers various execution cycles as specified in the following and as see in Figure 4.6).

The experiment design steps are:

- the tests were executed multiple times, throughout 8 cycles
- each cycle is a single execution of a subset of tests
- for the first 3 cycles we have executed all tests, without using MixTCP
- starting with the fourth cycle, we have started using MixTCP, that prioritized the tests by analyzing data from all previous cycles, thus, we conducted only a select subset of these tests, choosing them based on their assigned priority order
- in order to compute NAPFD we take the next cycle's test runs to decide each test file's verdict
- NAPFD is computed for cycles between 3 and 7, the 3rd cycle's NAPFD will tell how well the tests were predicted in the 4th cycle

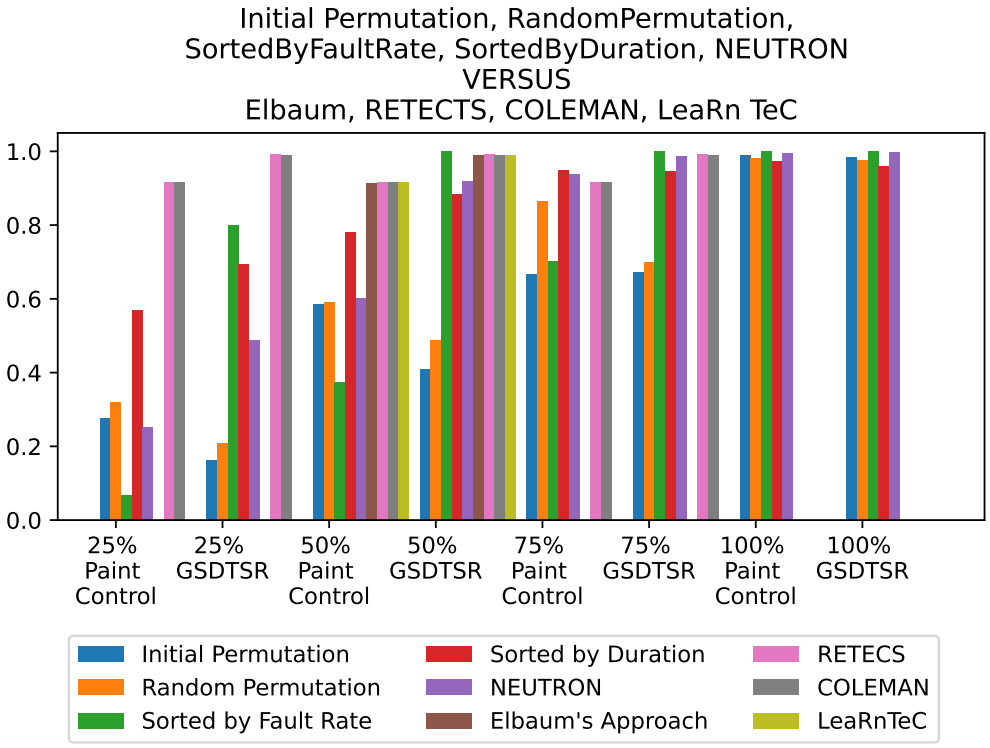

Figure 4.4: Experiment 1 with approximately 11x30 cycles, considering all budgets

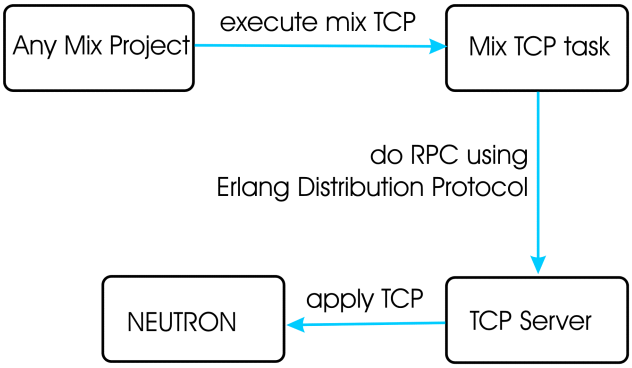

Figure 4.5: Overview of the MixTCP approach

- as there are only 8 cycles, no score can be computed for the 8th cycle yet
- there were 8 faults exposed throughout the experiment

In order to measure the results, we used the same metric as in the NEUTRON (from Section 4.1) approach, namely NAPFD. This metric provides a sufficient overview of the general performance of a TCP approach. The results are shown in Table 4.2. As one can observe, the performance of the solution is considerably better than running the tests in a Random way (which is the default behavior in Elixir).

Figure 4.7 graphically illustrates the outcomes for Random and NEUTRON across the cycles, clearly showing a notable enhancement in performance for both.

More details on this experiment can be found in the thesis.

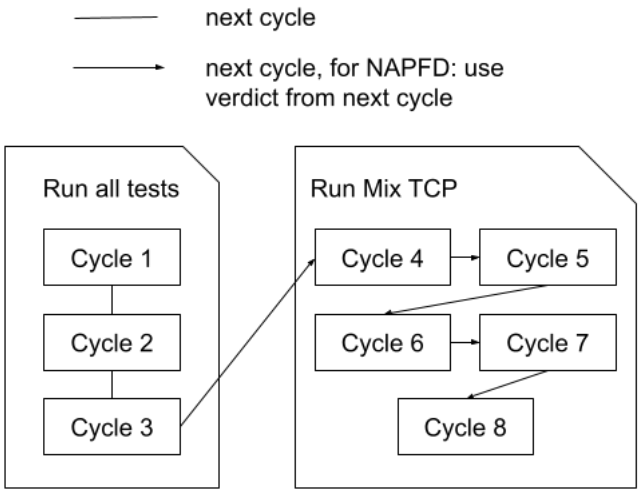

Figure 4.6: Experiment Cycles

Table 4.2: Random and NEUTRON NAPFD per cycle

Cycle	Random NAPFD	NEUTRON NAPFD
3	5.82%	12.18%
4	13.15%	22.25%
5	18.81%	35.06%
6	32.16%	48.20%
7	45.86%	81.46%

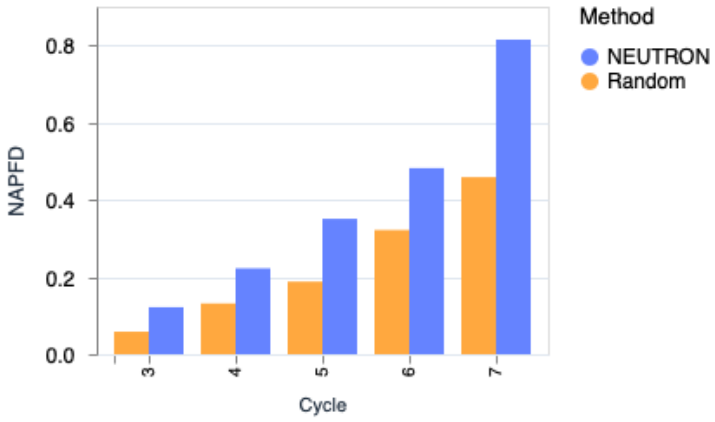

Figure 4.7: NAPFD per cycle

4.3 Conclusions and Further Work

In continuous integration settings, a set of tests is run for each new feature or bug fix. There are several strategies to select or prioritize the execution of the tests. Test Case Prioritization reorders the test cases so that faults are found earlier with a minimum execution cost.

This chapter investigated the use of neural network-based classification models to help prioritize tests, as well as the integration of it into MixTCP, an applied solution for real-world projects. Three different models were employed with various features (duration, fault rate, cycles count, total runs count) and considering information at every 30 cycles or at every 100 cycles.

The NEUTRON approach finds a better prioritization with respect to NAPFD than random permutation. The NEUTRON results are comparable with other sort-based solutions that used either duration or faults; in addition, it considers both features when constructing the solution. Compared to other existing state-of-the-art approaches, NEUTRON achieves similar competitive results when considering a budget of 50% and the best results when considering budgets of 75% and 100%.

The evaluation of MixTCP in real-world industrial settings validates its benefits in improving the TCP process. It also validates the theoretical approach, NEUTRON. The solution has shown a clear improvement in the prioritization of the test cases, which ultimately leads to earlier fault detection and also more efficient use of testing resources. This has significant implications for the speed and quality of software development, as it enables quicker releases and more reliable software products.

Our future efforts will be directed towards further experiments that are going to be performed with more projects and with more complex scenarios considering various cycles and more test cases discovering the same faults, along with the connection to the change in the source code or in the requirements. In addition, we would like to ensure that MixTCP successfully integrates in any development environment, we also plan to integrate other state-of-the-art TCP approaches into it, facilitating faster adoption by the software industry.

Chapter 5

Rough Sets Clustering for Test Case Prioritization

This chapter presents the application of our novel clustering algorithm in the context of test case prioritization and proposes a modern, unified approach to test case prioritization validated on a synthetic dataset using our clustering approach.

Section 5.1 applies our rough clustering method on industrial datasets in the context of test case prioritization in continuous integration.

Section 5.2 illustrates a novel, original idea regarding test case prioritization that unites all the well-known concepts into a single dataset. Moreover, it validates the efficacy of this new approach using our clustering approach.

Section 5.3 concludes the chapter and describes our future work.

The approaches introduced in this chapter represent original research contributions published in [5, 20].

5.1 RoughTCP: an Approach on TCP in CI Based on Rough Sets Clustering

The content of this section is derived from the original paper [5].

In the rapidly evolving landscape of Continuous Integration (CI), test case execution becomes pivotal with every code modification, making regression testing strategies indispensable. Among these, Test Case Prioritization (TCP) has become a popular way to improve the efficiency and effectiveness of software testing. Recently, researchers have been mostly looking at supervised learning methods, such as neural networks, to deal with TCP in CI. However, because of the dynamic nature of these environments, it might be worth exploring unsupervised approaches that can adapt to the inherent uncertainties without labeled data. This section proposes RoughTCP, a novel approach that utilizes a rough sets-based agglomerative clustering algorithm to prioritize test cases.

RoughTCP automatically groups and ranks tests based on their intrinsic patterns and correlations (e.g., faults, tests duration, cycles count, and total runs count) without a predefined model. This improves fault detection without the need for constant supervision and provides a more comprehensive understanding of the results by incorporating rough sets. Three sets of experiments were performed, considering data from continuous integration contexts in industrial projects.

Compared to recent related work, our experiments show that the RoughTCP approach yields better results for budgets higher than or equal to 75% on all datasets, while sometimes also outperforming all other methods on lower budgets. This underlines the potential of unsupervised methods and, in particular, the strength of RoughTCP in reshaping the TCP landscape in CI environments.

The section starts by presenting the test case prioritization problem and context (which is the

same as in Section 4.1) alongside state-of-the-art solutions. Next, it outlines our *RoughTCP* approach (as seens in Figures 5.1 and 5.2). The section continues with detailing the rough clustering algorithm that we used for TCP. More details can be found in the thesis.

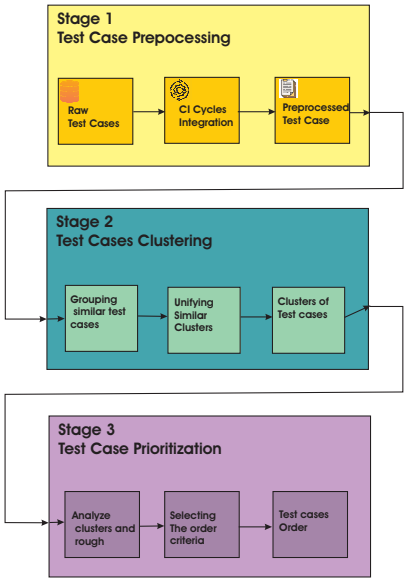

Figure 5.1: Approach stages

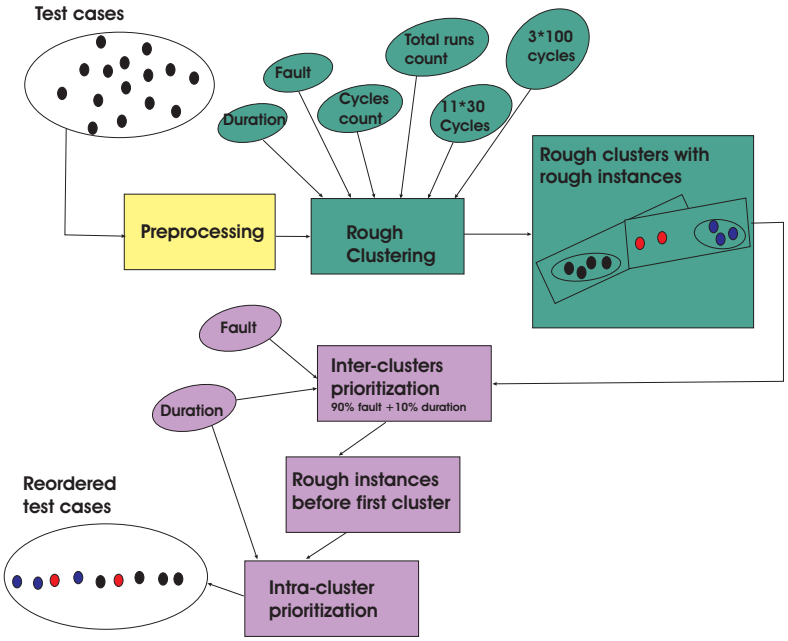

Figure 5.2: Approach overview

The results of the experiments performed using rough clustering are provided in Table 5.1, together with the results of other approaches from Elbaum's approach [2], to RETECS [13], COLEMAN [8], and NEUTRON.

Our investigations employing the experiments based on the Rough Clustering (RC) are: RC-Reduced (using the 4 features discussed above), RC-3*100-cycles (using additional 3 features based on the three 100 cycles), and RC-11*30-cycles (using additional eleven features based on the eleven 30 cycles). More experiments can be found in the thesis.

Table 5.1: Experiments

Project	Budget	NAPFD								
		Elbaum	**RETECS RNFail**	**RETECS TimeRank**	**COLEMAN RNFail**	**COLEMAN TimeRank**	**NEUTRON**	**RC-Reduced**	**RC-3x100-cycles**	**RC-11x30-cycles**
Paint Control	10		**0.9078**	**0.9077**	**0.9076**	**0.9076**		0.1605	0.0393	0.0421
	25						0.2512	**0.5104**	0.4877	0.4767
	50	**0.9145**	**0.915**	0.9138	**0.915**	**0.915**	0.6004	0.7579	0.7352	0.7354
	75						0.9377	**0.9490**	0.9264	0.9265
	80		0.9162	0.9160	0.9171	0.9171		**0.9827**	0.9264	0.9265
	100						0.9940	0.9940	**0.9946**	**0.9947**
GSDTSR	10		**0.9893**	**0.9893**	**0.9894**	**0.9894**		0.8904	0.8904	0.8969
	25						0.4868	**0.9533**	**0.9533**	**0.9551**
	50	0.9891	**0.9911**	0.9906	0.9893	0.9894	0.9175	0.9880	0.9880	0.9880
	75						0.9870	**0.9967**	**0.9967**	**0.9965**
	80		0.9921	0.9914	0.9893	0.9894		**0.9977**	**0.9976**	**0.9974**
	100						0.9983	**0.9992**	**0.9992**	**0.9990**
IOF/ROL	10		**0.3704**	**0.3779**	0.3632	0.3670		0.1578	0.1004	0.1939
	25							0.4330	0.3802	**0.4718**
	50	0.4892	0.5101	0.5025	0.5046	0.5189		**0.7909**	0.7762	**0.8125**
	75							0.8784	**0.9019**	**0.8964**
	80		0.5495	0.5287	0.5569	0.5678		0.8874	**0.9107**	**0.9026**
	100							0.9180	**0.9318**	0.9199

5.2 Embracing Unification: a Comprehensive Approach to Modern Test Case Prioritization

The content of this section is derived from the original paper [20].

Regression testing is essential for software systems that undergo changes to ensure functionality and identify potential problems. It is crucial to verify that modifications, such as bug fixes or improvements, do not affect existing functional components of the system. Test Case Prioritization (TCP) is a strategy used in regression testing that involves the reordering of test cases to detect faults early on with minimal execution cost.

Current TCP methods have investigated various approaches, including source code-based coverage criteria, risk-based, and requirement-based conditions. However, to our knowledge, there is currently no comprehensive TCP representation that effectively integrates all these influencing aspects. Our approach aims to fill this gap by proposing a comprehensive perspective of the TCP problem that integrates numerous aspects into a unified framework: traceability information, context, and feature information.

To validate our approach, we use a synthetic dataset that illustrates six scenarios, each with varying combinations of test cases, faults, requirements, execution cycles, and source code information. Three methods, Random, Greedy, and Clustering, are employed to compare the results obtained under various time-executing budgets. Experiment results show that the Clustering method consistently outperforms Random and Greedy across various scenarios and budgets.

The section begins with outlining the context of the investigation, along with with the motivation, TCP background, and state-of-the-art approaches. Then we present our novel, unified approach to modern TCP (also seen on Figure 5.3). Finally, we provide the case study with the synthetic dataset and the 6 scenarios for building TCP solutions. More details can be found in the thesis.

The metric used for the evaluation is the NAPFD [10], however, for each scenario, different information is incorporated into the formula. For example, in the requirements-based scenario,

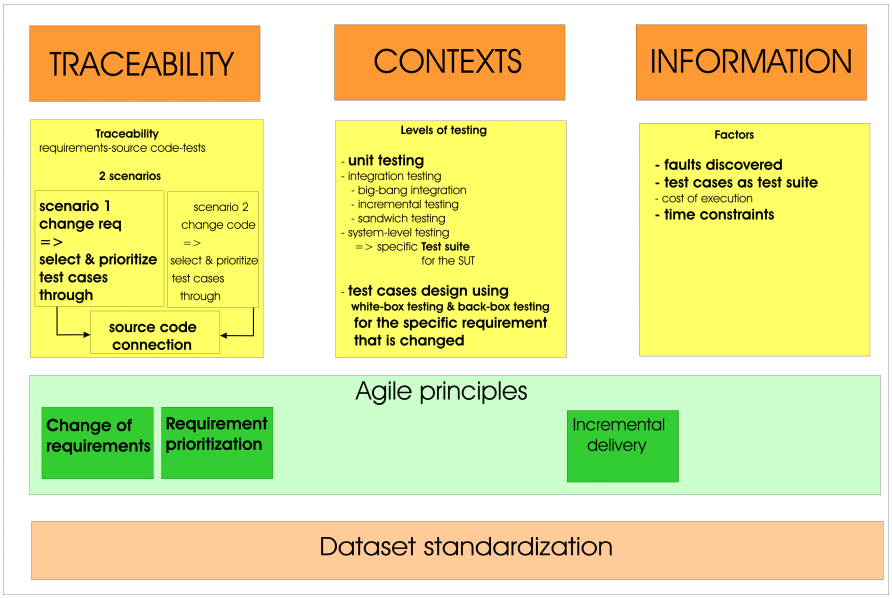

Figure 5.3: Overview of the approach

the faults that are considered are those linked to the requirements that are changed.

The TCP solutions obtained for each employed method are provided in Table 5.2. The experiments were performed with various time budget configurations, from 10 to 100. The majority of the best results obtained are for the Clustering method for almost all budgets. The best results are in bold in Table 5.2.

Table 5.2: NAPFD values for the Random, Greedy and Clustering approaches for various budget values.

Scenario	S1						S2						S3					
Budget	10	25	50	75	80	100	10	25	50	75	80	100	10	25	50	75	80	100
Random	13.54	25.25	39.29	51.8	53.93	64.46	8.64	13.47	21.2	28.16	29.59	36.04	9.95	17.78	27.15	35.51	36.96	43.53
Greedy	21.43	35.71	64.29	**77.27**	79.17	**85.29**	**21.43**	**28.57**	**47.62**	50.79	51.43	53.78	**28.57**	**42.86**	56.12	60.71	61.69	**65.13**
Clustering	**42.86**	**57.14**	**72.22**	**77.27**	**80.77**	**85.29**	**21.43**	**28.57**	45.71	**51.95**	**52.75**	**53.78**	**28.57**	**42.86**	**58.04**	**62.5**	**63.19**	**65.13**
Scenario	S4						S5						S6					
Budget	10	25	50	75	80	100	10	25	50	75	80	100	10	25	50	75	80	100
Random	8.29	15.77	24.62	32.48	33.73	39.1	5.7	9.91	17.74	24.25	25.66	32.01	3.48	6.54	11.94	17.1	18.05	23.14
Greedy	21.43	**42.86**	**50**	**51.95**	**52.38**	**53.78**	21.43	**42.86**	50	51.95	52.38	**53.78**	**21.43**	37.5	40.18	40.91	41.07	**41.6**
Clustering	**37.5**	**42.86**	**50**	**52.04**	52.04	52.94	**28.57**	**42.86**	**51.43**	**53.06**	**53.06**	**53.78**	**21.43**	**38.57**	**40.71**	**41.21**	**41.33**	**41.6**

The obtained solutions for Greedy and Clustering are provided at this link along with the dataset [18].

The Greedy solution is: [10, 11, 5, 1, 9, 12, 13, 6, 3, 2, 15, 16, 7, 17, 4, 8, 14]. For the Clustering solution, the clusters are also emphasized: [1, 5], [11], [10, 12], [13, 9], [15], [14, 4, 8, 17, 7], [6, 16, 2, 3]. The analysis reveals that while the Greedy solution identifies the correct test cases sooner than the Clustering solution, it overlooks the duration of these tests. In contrast, the Clustering solution incorporates all relevant information, including test duration, thereby offering a more comprehensive and balanced solution. More information about the results can be found in the thesis.

5.3 Conclusions and Further Work

In regression testing, Test Case Prioritization is one of the strategies to be used, which aims to order the test cases based on different, well-defined criteria. The proposed approach, RoughTCP, uses a rough set clustering algorithm to group and rank test cases based on their intrinsic patterns

in information about faults, duration, and continuous integration cycles.

The approach was rigorously validated using three distinct industrial projects, each providing information from their continuous integration environments, such as duration, faults, or information from up to 350 cycles. Overall results show that RoughTCP outperforms the current state-of-the-art solutions.

This chapter also proposed a comprehensive unification of different features and data that may be needed during a TCP process. As discussed, currently, there is no formal definition of TCP that adequately covers the various influencing factors. From our case study and validation, we can empirically state that having comprehensive datasets and a complete solution for them enhances the overall performance and precision of the process.

Further work may consider other projects to validate RoughTCP on. The characteristics of the projects can also be taken into account during rough clustering. Another aspect that is important in the regression testing context is related to the traceability between requirements and test cases, thus features that encapsulate it may be of interest for future investigations. For the unification, our vision is to fill the missing gap with a unified TCP framework that adequately integrates numerous perspectives: requirements, context, and code information. The framework should be able to assist both with building datasets and solutions for test case prioritization.

Chapter 6

Conclusions and Future Work

This thesis presented our research on theoretical machine learning algorithms as well as their application in different contexts, including software testing.

First of all, let us address all the objectives mentioned in Chapter 1. We have started with Objective 1 and studied both fuzzy and rough set theories as both are suitable for uncertain contexts, we have settled on applying rough sets, because they are easier to apply and adapt to all types of data. Objective 2 and Objective 3 were achieved by designing and implementing ABARC (described in Section 3.1). Objective 4, Objective 5, and Objective 6 are all about the evaluation of our new rough sets based clustering methods, we have used standard methodologies for the usual evaluation of the clustering results, but for rough clustering results we have not found any existing evaluation methodology so we have come up with a novel methodology and implemented it to evaluate results generated by rough clustering methods (found in Chapter 3). Objective 7 was carried out by applying our rough clustering methods on various standard datasets and industrial datasets from software testing (explained in Section 5.1) as well as on other datasets from other domains. Objective 8 specifies the research of a new domain, software testing, in which our novel methods turned out to be useful. We have researched this field, and have found usages for our proposed methods for test case prioritization (TCP). Objective 9 describes our way of contributing to the TCP field from a more theoretical perspective, we have built a novel and unified approach for TCP that would combine all of the information known to provide a more accurate regression testing (more details can be found in Section 5.2). Objective 10 is also about researching the field of TCP, more specifically about analyzing state-of-the-art learning methods. After our analysis, we have proposed new neural network-based models for TCP as mentioned in Objective 11 (more information can be found in Section 4.1). Objective 12 was achieved by building the MixTCP solution in the context of TCP in CI (described in Section 4.2). Finally, Objective 13 mentions the application of our new rough clustering approaches in software testing and was achieved by applying them on industrial datasets in the context of TCP (Section 5.1) and using them for an experiment in the framework of the unified approach mentioned above (Section 5.2).

We have developed innovative agglomerative clustering methods that employ rough set theory to identify hybrid data, including outliers and rough instances that exhibit characteristics aligning with multiple clusters. This approach enhances data analysis by providing deeper insights into complex datasets, potentially revealing new classes or anomalies. Our algorithm's scalability is ensured by software agents that enable efficient computation on modern computers, even for large datasets. Comparative analysis with other classification techniques has shown our algorithm to outperform most in terms of accuracy, entropy, and the Adjusted Rand Index, even more so after outlier elimination.

In addition, we introduced an objective methodology for evaluating the quality of the identified hybrid data, offering a more reliable assessment compared to traditional cluster quality measures. This new validation methodology has proven effective in our comparative analysis, demonstrating our approach's robust performance.

Our research also investigates the impact of various similarity measures on clustering outcomes, especially concerning rough instances. Given the challenge posed by the lack of specific information on overlapping regions in standard datasets, we proposed a novel methodology for identifying potential rough instances, which could serve as a benchmarking tool.

We have also conducted a comprehensive evaluation of our clustering results, including the rough ones. We have considered internal, external and rough evaluation metrics too. The comparison with related work shows that our methods are the best in most situations.

In the context of continuous integration, automated tests are typically rerun to ensure correctness whenever a new feature is implemented or a bug fixed. However executing all tests can be time-consuming and resource-intensive, thus choosing the right strategy for Test Case Prioritization (TCP) is crucial to detect faults early and efficiently.

Our investigation into neural network-based models, specifically the NEUTRON approach, shows promising prioritization capabilities, rivaling state-of-the-art methods by considering both test duration and fault rates. MixTCP's real-world application has significantly improved the TCP process, leading to faster fault detection and more efficient resource utilization, which in turn accelerates software release times and enhances product reliability.

Furthermore, we proposed the RoughTCP approach, which employs our rough set theory-based clustering methods for TCP in regression testing. The methods prioritize test cases based on their patterns related to faults, duration, and continuous integration cycles. Validated on three industrial projects, RoughTCP outperforms existing solutions in most of the cases.

Our work also emphasizes the need for a unified approach to TCP, integrating various data and features to improve the process' effectiveness. This comprehensive strategy, validated through case studies, shows improvement in TCP's performance and accuracy, highlighting the importance of a complete and versatile solution for enhancing software development quality and efficiency.

Regarding future directions, first we want to conduct additional experiments with datasets featuring more extensive overlapping regions and clusters of varied shapes. We aim to explore the effect of outliers on our findings and examine the influence of diverse distance metrics on these outcomes.

Next, we plan to extend our research on NEUTRON and RoughTCP by applying these methods to a broader range of projects and more intricate scenarios that include various cycles and an increased number of test cases that identify identical faults. This also involves examining the relationship between changes in source code or requirements and test case outcomes. Moreover, we intend to enhance MixTCP's adaptability across different development environments by incorporating various advanced TCP methodologies as well as extending it to multiple programming languages.

Lastly, our future research will potentially include an evaluation of project-specific characteristics that could influence rough clustering outcomes. In the context of regression testing, the ability to trace the connection between requirements and test cases is crucial, suggesting that features capturing this relationship could be valuable for further study. Our ultimate objective is to

develop a comprehensive TCP framework that seamlessly integrates multiple dimensions, like requirements, context, and code information, thus bridging the existing gap in the field. This framework would support the creation of datasets and the development of solutions for test case prioritization, offering a holistic approach to improving software testing processes.

Bibliography

[1] ALPAYDIN, E. *Machine learning*. MIT press, 2021.

[2] ELBAUM, S., ROTHERMEL, G., AND PENIX, J. Techniques for Improving Regression Testing in Continuous Integration Development Environments. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (New York, NY, USA, 2014), FSE 2014, Association for Computing Machinery, p. 235–245.

[3] GARCIA-DIAS, R., VIEIRA, S., PINAYA, W. H. L., AND MECHELLI, A. Clustering analysis. In *machine learning*. Elsevier, 2020, pp. 227–247.

[4] GAROUSI, V., RAINER, A., LAUVÅS JR, P., AND ARCURI, A. Software-testing education: A systematic literature mapping. *Journal of Systems and Software 165* (2020), 110570.

[5] GĂCEANU, R. D., SZEDERJESI-DRAGOMIR, A., AND VESCAN, A. Leveraging Rough Sets for Enhanced Test Case Prioritization in a Continuous Integration Context. In *7th Workshop on Validation, Analysis and Evolution of Software Tests (VST 2024), at the 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER 2024)* (Rovaniemi, Finland, 12 march 2024), IEEE. *accepted for publication*.

[6] GĂCEANU, R. D., SZEDERJESI-DRAGOMIR, A., POP, H. F., AND SÂRBU, C. ABARC: An agent-based rough sets clustering algorithm. *Intelligent Systems with Applications 16* (2022), 200117.

[7] KHATIBSYARBINI, M., ISA, M. A., JAWAWI, D. N., AND TUMENG, R. Test case prioritization approaches in regression testing: A systematic literature review. *Information and Software Technology 93* (2018), 74–93.

[8] LIMA, J. A. P., AND VERGILIO, S. R. A Multi-Armed Bandit Approach for Test Case Prioritization in Continuous Integration Environments. *IEEE Transactions on Software Engineering 48*, 2 (2022), 453–465.

[9] OMRI, S., AND SINZ, C. Learning to Rank for Test Case Prioritization. In *2022 IEEE/ACM 15th International Workshop on Search-Based Software Testing (SBST)* (2022), pp. 16–24.

[10] QU, X., COHEN, M., AND WOOLF, K. Combinatorial Interaction Regression Testing: A Study of Test Case Generation and Prioritization. In *2007 IEEE International Conference on Software Maintenance* (Los Alamitos, CA, USA, oct 2007), IEEE Computer Society.

[11] SHAHIN, M., BABAR, M. A., AND ZHU, L. Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. *IEEE access 5* (2017), 3909–3943.

[12] SHARMA, R., SHARMA, K., AND KHANNA, A. Study of supervised learning and unsupervised learning. *International Journal for Research in Applied Science and Engineering Technology 8*, 6 (2020), 588–593.

[13] SPIEKER, H., GOTLIEB, A., MARIJAN, D., AND MOSSIGE, M. Reinforcement Learning for Automatic Test Case Prioritization and Selection in Continuous Integration. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis* (New York, NY, USA, 2017), ISSTA 2017, Association for Computing Machinery, p. 12–22.

[14] SZEDERJESI-DRAGOMIR, A. A Comprehensive Evaluation of Rough Sets Clustering in Uncertainty Driven Contexts. *Studia Universitatis Babeș-Bolyai Informatica 69*, 1 (2024), 41–56.

[15] SZEDERJESI-DRAGOMIR, A., GĂCEANU, R. D., POP, H. F., AND SÂRBU, C. A Comparison Study of Similarity Measures in Rough Sets Clustering. In *Proceedings of the 2019 IEEE 15th International Scientific Conference on Informatics (INFORMATICS 2019). Editors: William Steingartner, Štefan Korečko, Anikó Szakál* (Poprad, Slovakia, Nov 20 – 22, 2019), IEEE Press, pp. 399 – 405.

[16] SZEDERJESI-DRAGOMIR, A., GĂCEANU, R. D., POP, H. F., AND SÂRBU, C. Experiments on Rough Sets Clustering with Various Similarity Measures. *IPSI BGD TRANSACTIONS ON INTERNET RESEARCH 16* (2020), 75–83.

[17] SZEDERJESI-DRAGOMIR., A., GĂCEANU., R., AND VESCAN., A. Industrial Validation of a Neural Network Model Using the Novel MixTCP Tool. In *Proceedings of the 19th International Conference on Evaluation of Novel Approaches to Software Engineering - ENASE 2024* (Angers, France, 29 april 2024), INSTICC, SciTePress, pp. 110–119.

[18] VESCAN, A., GACEANU, R., AND SZEDERJESI-DRAGOMIR, A. Dataset and Results for Embracing Unification: A Comprehensive Approach to Modern Testcase Prioritization. `https://figshare.com/s/250342cc522867910bc4`, Accessed: 2024-03-09.

[19] VESCAN, A., GĂCEANU, R. D., AND SZEDERJESI-DRAGOMIR, A. Neural Network-Based Test Case Prioritization in Continuous Integration. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)* (sep 2023), IEEE, IEEE Computer Society, pp. 68–77.

[20] VESCAN., A., GĂCEANU., R., AND SZEDERJESI-DRAGOMIR., A. Embracing Unification: a Comprehensive Approach to Modern Test Case Prioritization. In *Proceedings of the 19th International Conference on Evaluation of Novel Approaches to Software Engineering - ENASE 2024* (Angers, France, 29 april 2024), INSTICC, SciTePress, pp. 396–405.

[21] YOO, S., AND HARMAN, M. Regression testing minimization, selection and prioritization: a survey. *Software testing, verification and reliability 22*, 2 (2012), 67–120.

[22] Mix. `https://hexdocs.pm/mix/Mix.html`. Accessed: 2024-05-15.