

Cross-Project Code Smell Detection as a Dynamic Optimization Problem: An Evolutionary Memetic Approach

1st Sofien Boutaib

SMART lab, University of Tunis, ISG
Tunis, Tunisia
mohamedsofien.boutaib@isg.u-tunis.tn

2nd Maha Elarbi

SMART lab, University of Tunis, ISG
Tunis, Tunisia
arbi.maha@yahoo.com

3rd Slim Bechikh

SMART lab, University of Tunis, ISG
Tunis, Tunisia
slim.bechikh@fsegn.rnu.tn

4th Carlos A. Coello Coello

Departamento de Computación, CINVESTAV-IPN
Mexico City, Mexico
ccoello@cs.cinvestav.mx

5th Lamjed Ben Said

SMART lab, University of Tunis, ISG
Tunis, Tunisia
bensaid_lamjed@yahoo.fr

Abstract—Code smells signal poor software design that can prevent maintainability and scalability. Identifying code smells is difficult because of the large volume of code, considerable detection expenses, and the substantial effort needed for manual tagging. Although current techniques perform well in within-project situations, they frequently struggle to adapt to cross-project environments that have varying data distributions. In this paper, we introduce CLADES (Cross-project Learning and Adaptation for Detection of Code Smells), a hybrid evolutionary approach consisting of three main modules: Initialization, Evolution, and Adaptation. The first module generates an initial population of decision tree detectors using labeled within-project data and evaluates their quality through fitness functions based on structural code metrics. The evolution module applies genetic operators (selection, crossover, and mutation) to create new offspring solutions. To handle cross-project scenarios, the adaptation module employs a clustering-based instance selection technique that identifies representative instances from new projects, which are added to the dataset and used to repair the decision trees through simulated annealing. These locally refined decision trees are then evolved using a genetic algorithm, thus enabling continuous adaptation to new project instances. The resulting optimized decision tree detectors are then employed to predict labels for the new unlabeled project instances. We assess CLADES across five open-source projects and we show that it has a better performance with respect to baseline techniques in terms of weighted F1-score and AUC-PR metrics. These results emphasize its capacity to effectively adjust to different project environments, facilitating precise and scalable detection of code smells while minimizing the need for manual review, contributing to more robust and maintainable software systems.

Index Terms—Code smells, Evolutionary algorithms, Simulated annealing, Clustering-based instance selection, Cross-project, Software maintenance.

I. INTRODUCTION

Software maintenance and evolution require developers to identify and eliminate several types of issues, such as code

smells. This may hinder the quality and maintainability of the software. The detection of code smells requires analyzing the source code. Such task is challenging and time-consuming [1], especially for large projects with thousands of code files. Consequently, researchers have proposed a variety of automated techniques for detecting code smells, such as machine learning and search-based techniques [1]. Several existing within-project code smell detection techniques have been proposed in the literature. They can be classified into three categories: Rule-based, Machine learning-based, and Search-based approaches. Techniques in the first category (e.g., DECOR [2]) combine several metrics to identify different smell types using predefined rule-cards. In contrast, techniques in the second category (e.g., Decision Trees (DTs) [3] and deep neural networks) train classifiers using training sets to predict instances smell types. Techniques in the last category (e.g., genetic programming [4]) optimize a set of detectors to enhance the detection.

These methods work over within-project contexts. However, they may face limitations when applied to cross-project scenarios with disparate data distributions [5]. Within-project techniques typically train models on a single project. They are applied to similar projects for code smell detection. However, they may exhibit limited effectiveness. This is due to the fact that the resulting detectors often lack applicability to projects coming from different domains [6]. On the other hand, cross-project code smell detection techniques draw upon knowledge from multiple projects to enhance detection performance. Frequently, they encounter the problem of domain adaptation. This appears when models trained on one project struggle to generalize to projects coming from different domains [6]. Moreover, labeling an entire project for training purposes can be infeasible due to the associated high cost and effort.

In this paper, we present CLADES (Cross-project Learning and Adaptation for Detecting Code Smells), which is a novel

hybrid evolutionary optimization framework. The main goal of CLADES is to determine whether a software class is smelly or not. This could be done using a single label (i.e., smelly or not-smelly) for each class. Our proposed approach focuses on detecting complex code smells, such as Feature Envy and God Class. These smell types are challenging to define using simple rules. CLADES combines Genetic Algorithms (GAs) for constructing DTs with Simulated Annealing (SA) for refining these DTs. The GA is employed to evolve DTs from within-project data, optimizing their structure and feature selection to establish robust initial models. When a new project dataset is encountered, a clustering-based Representative Instance (RI) selection technique identifies a diverse subset of instances. These RIs are then used to refine the DTs through SA. This allows them to adapt to the characteristics of the new dataset. This hybrid approach ensures that DTs are continuously improved through iterative adaptation, enabling them to effectively handle the evolving distributions of the new project. Our experimental evaluation is conducted on five open-source projects. Such evaluation shows the superior performance of CLADES in terms of two evaluation metrics (i.e., weighted F1-score and AUC-PR) when compared with respect to existing techniques. The results highlight the effectiveness of integrating SA into the DTs repairing process. It achieves high detection accuracy while significantly reducing labeling effort. Overall, CLADES offers an adaptable and efficient solution for cross-project code smell detection. To sum up, the main contributions of this paper are:

- 1) Introducing CLADES, a novel hybrid evolutionary approach that integrates GAs with SA to enhance cross-project code smell detection;
- 2) Proposing a clustering-based Representative Instance (RI) selection technique to improve model adaptability by selecting diverse and informative code examples from new projects;
- 3) Developing an incremental adaptation mechanism that refines decision trees dynamically using SA, enabling continuous improvement based on evolving data distributions;
- 4) Showing the effectiveness of CLADES through extensive experimental evaluation on five open-source projects to illustrate its superior performance compared to existing rule-based and search-based approaches in terms of F1-score and AUC-PR.

II. BACKGROUND

Detecting code smells serves as an indicator of poor software design. Such task is a critical one in software maintenance. While numerous automated techniques exist, such as rule-based (e.g., DECOR [2]), machine learning-based (e.g., DTs [3]), and Search-based techniques (e.g., Multi-Objective Genetic Programming (MOGP) [4]), the latter have shown promising results within individual projects. Unfortunately, these techniques face problems when applied across different projects. Among these problems, we can find the discrepancies in data distributions. This appears when there are variations in

coding standards, architectures, and domain-specific practices. Consequently, the models trained on one project could face challenges to generalize to others [7]. Moreover, labeling datasets for training purposes is resource-intensive, further complicating the development of robust cross-project detection models. Adaptive approaches that incorporate instance selection and incremental refinement have been proposed to address these issues [8]. Within this realm, DT-based models have garnered considerable attention owing to their virtues of simplicity, interpretability, and their capacity to accommodate both numerical and categorical data. Remarkably, the C4.5 algorithm stands out as one of the most renowned DT algorithms, utilizing information gain to determine the optimal split at each node within the tree [9]. To enhance the DT structures, evolutionary algorithms have been applied (e.g., GAs). The latter optimize DT structures and feature selection for code smell detection. However, GAs may suffer from slow convergence. Therefore, their performance can be improved by applying local search (such as SA, Tabu search, etc.) [10]. To the best of our knowledge, selecting RIs from new cross-project data can be effectively achieved through clustering techniques, ensuring that the model adapts efficiently to diverse datasets. Clustering helps to identify subsets of data that are both informative and diverse, reducing the labeling effort while maintaining the robustness of the refined model. Combining clustering-based instance selection with DT optimization through the use of GAs and SA offers a promising direction for overcoming cross-project challenges in code smell detection.

CLADES was specifically designed to address the complexities of adapting to varying data distributions across projects by leveraging DTs as its core detectors. Initially, CLADES generates and evolves DTs using within-project datasets. To adapt these DTs to cross-project scenarios, our approach employs a clustering technique to identify a diverse and representative subset of instances from new projects. These RIs are subsequently utilized to fine-tune the DTs using SA. This fine-tuning process iteratively adjusts the DTs' splits and structures to effectively incorporate the unique characteristics of the new project. Therefore, the DTs are refined based on an optimization process that expands their leaf nodes or adjusts their structure to encompass the new Java project RIs.

III. CROSS-PROJECT LEARNING FOR AUTOMATED DETECTION OF CODE SMELLS (CLADES)

In this section, we describe our proposed approach CLADES. Fig. 1 provides an overview of the three main steps of our approach: (1) Initialization Module, (2) Evolution Module, and (3) Adaptation Module. The first step is performed to generate an initial population of DT detectors using labeled data from a set of within-projects (as described in Table I). These DTs are designed to learn and predict code smells based on structural code metrics. Moreover, a fitness evaluation step is applied on all the generated DTs to evaluate their quality. In the second step, we apply the evolutionary operators (i.e., selection, crossover, and mutation) to create new offspring solutions. Once a new project is received, we

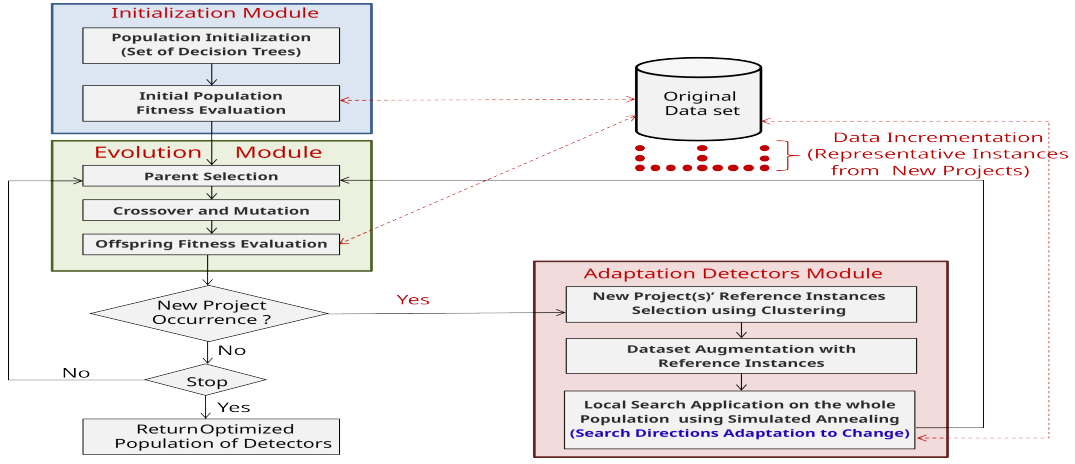


Fig. 1. Illustration of the global framework of CLADES.

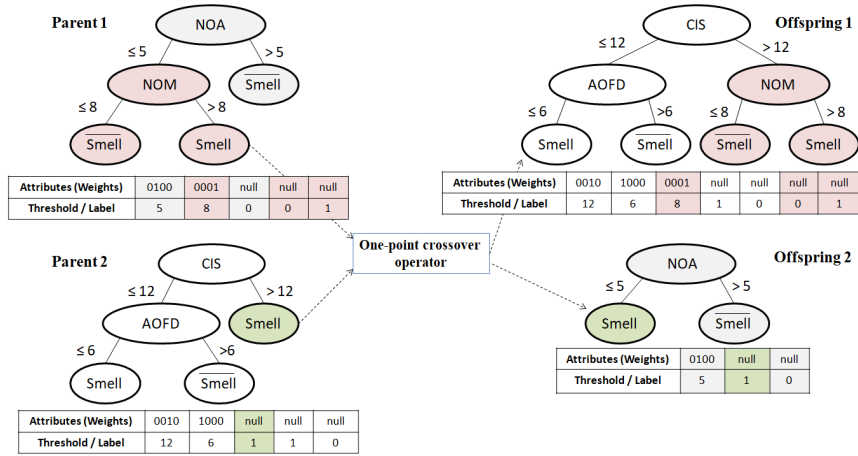


Fig. 2. An illustrative example of the one-point crossover operator used in CLADES.

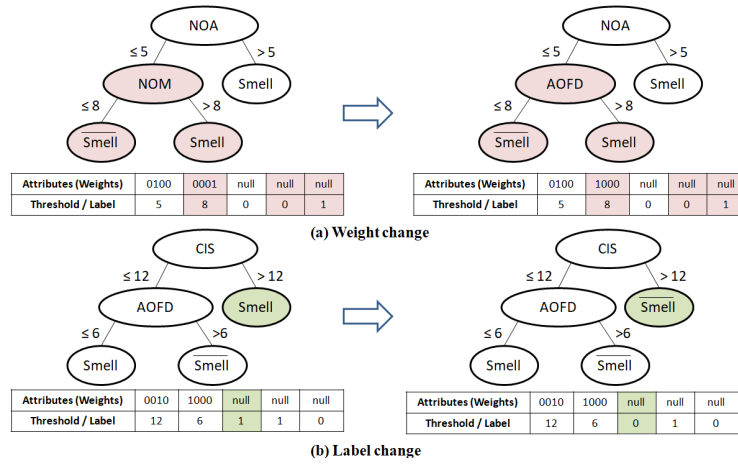


Fig. 3. An illustrative example of the (a) Weight change and (b) Label change mutation operators adopted in CLADES.

apply the adaptive model which is composed by three steps that can be described as follows. The first step utilizes a clustering approach to select RIs from the new project. These selected RIs are added to the data set and then they are used by SA which locally refines the DTs to allow the detectors to incorporate information extracted from the RIs of the new project so that it can adapt to changes in data distribution. The combination of the GA (global search) and SA (local search) with the continuous adaptation of new cross-project instances enables CLADES to be an efficient and accurate tool of code smells detection. This process is repeated till the stopping criterion is reached.

A. Initialization Module

The Initialization Module is responsible for generating the initial set of code smell detectors. This process begins with the initialization of the population, where a diverse set of candidate DT models is created. To the best of our knowledge, each DT is encoded as a chromosome, utilizing a two-array structure. The first array represents decision nodes. This array is encoded using binary vectors that contain weights for different structural code metrics (such as lines of code (LOC), cyclomatic complexity (CC), method count (MC), field count (FC), coupling between objects (CBO), lack of cohesion (LCOM), depth of inheritance tree (DIT), and number of children (NOC)). In contrast, the second array contains corresponding threshold values used for splitting data at each decision node. Terminal nodes in the DTs are represented by NULL values in the first array. However, they are represented by a 0 or 1 in the second array to indicate the absence or presence of a code smell, respectively. Such individual encoding provides a flexible and comprehensive structural representation of the DT. Following the initialization, we proceed with the Initial Population Fitness Evaluation. Each DT is evaluated using a fitness function defined as a weighted F1-score. The F1-score is specifically weighted with a β value set to 0.5, placing more emphasis on recall due to the imbalanced nature of code smell datasets. The F1-score is calculated as follows:

$$F_{\beta} = \frac{(1 + \beta^2) \cdot (\text{precision} \cdot \text{recall})}{(\beta^2 \cdot \text{precision} + \text{recall})}, \quad (1)$$

$$\text{precision} = \frac{TP}{TP + FP},$$

$$\text{recall} = \frac{TP}{TP + FN}.$$

where TP , FP , and FN are true positives, false positives, and false negatives, respectively. This evaluation prioritizes the accurate detection of code smells by minimizing false negatives and, therefore, ensures that the search starts with a good population of detectors. It is important to note that the initialization model is executed only once during the whole process since it serves only to generate the initial population.

B. Evolution Module

The evolution module is conceived to apply the genetic operators on the generated detectors. The selection operator

selects the fittest individuals from the population based on their fitness scores, which are calculated using the F1-score metric. CLADES uses a binary tournament selection approach, where two individuals are randomly selected, and the fittest one is selected for the next generation. For the crossover operator, we employ the standard one-point crossover operator, where a random point in the chromosomes is selected, and the genetic material beyond that point is swapped between the parents (cf. Fig. 2). We also adopt two mutation operators: (1) the sub-tree mutation operator (weight change) and (2) the leaf mutation one (label change). The former selects a random sub-tree from the tree and replaces it with a new randomly generated sub-tree, while the latter selects a terminal node and replaces its class prediction with the opposite one. Fig. 3 illustrates the two mutation operators adopted in CLADES.

C. Adaptation Detectors Module

This module is activated each time a new project appears. The main objective of this module is to deal with the challenges existing in cross-project code smell detection. The process begins with the selection of RIs from a new project using clustering. The K-means clustering technique is employed to select the RIs that belong to a new project. The number of clusters K of the K-means clustering model is determined dynamically using the Elbow method [11]. This method identifies the optimal K value for each project by analyzing the within-cluster sum of squares and selecting the point where additional clusters yield diminishing returns, ensuring adaptability to the specific characteristics of each dataset. This clustering step groups similar code classes based on a dissimilarity metric (e.g., Euclidean distance). This ensures the encompassing of the inherent diversity of the target data. To the best of our knowledge, one-third of the instances from each identified cluster for labeling are selected. This strategy is adopted to catch the diverse nature of code smells in the project and minimizes the effort required for labeling. These RIs are chosen based on a combination of their proximity to the cluster's centroid. Instances that are both centrally located within their respective clusters and display good classification results are prioritized for labeling. This way, only instances that are both informative and representative of the underlying data distribution are selected. Then, the process moves to Dataset Increase with Reference Instances. These selected RIs are labeled based on a collection of rule-based code smell detection tools such as DECOR, PMD, JDeodorant, etc. It is crucial to have human experts for the confirmation of the tools results and to have highly reliable labels for further model adaptation. The labeled RIs are then added to the existing labeled dataset. Finally, the core of adaptation involves local search using SA. The labeled RIs are now used by the SA algorithm to refine the existing DT detectors obtained from the evolution module. SA iteratively explores adjustments to each DT by modifying split points, feature weights, and sub-tree structures. The refinement process begins with a high initial temperature, which allows SA to make large structural changes and explore the search space extensively. As the temperature

decreases, the modifications become more subtle. This allows to converge towards a more refined DT structure. This adaptation ensures that the DTs incorporate the information from the new RIs, allowing them to effectively handle the characteristics of new projects. SA starts by an initialization step in which we determine its main parameters. Then, a new neighbor candidate solution DT' is generated using the same principle of our employed mutation operator. Next, the fitness function of the DT' solution is calculated. After that, we compare the fitness functions of the current solution DT_i and the new candidate solution DT' . In fact, two scenarios can appear. In the case where F' is superior to F_i , we accept a new candidate solution, while in the case where F_i is better than F' , the acceptance of DT' depends on a probability value P . The acceptance probability is calculated as:

$$P = \exp\left(-\frac{\Delta E}{T}\right) \quad (2)$$

where ΔE is the cost difference between the new and current solution. As the temperature T decreases, P decreases, shifting the algorithm from exploration to exploitation. During the process, the temperature parameter T_i is updated until a stopping criterion is reached.

D. Code Smell Prediction

After generating the set of optimized detectors that are created by our CLADES, we perform the code smell prediction task. The optimized DTs are employed to predict the label of each unlabeled instance. Each of these detectors will predict the labels of the unlabeled instances. Once predictions are made using all the detectors, a majority vote technique is applied to determine the final label of each instance. In other words, the predicted labels are compared, and the label with the highest number of votes is selected as the final predicted label for the instance. This approach helps to ensure that the final predicted labels are robust and reliable, as they are based on the combined predictions of multiple updated DT detectors.

IV. EXPERIMENTAL SETUP

Our study aimed to comprehensively evaluate CLADES for cross-project code smell detection. This evaluation serves the interests of both researchers and practitioners. The evaluation is guided by two main Research Questions (RQs):

- **RQ1:** *How does the performance of several code smell detection methods vary when trained on within-projects and tested on cross-projects?*
- **RQ2:** *Does CLADES outperform existing methods in handling diverse cross-project data distributions?*

A. Context of the study

In this study, we selected a diverse set of within- and cross-projects, as detailed in Tables I and II, based on criteria such as source code availability, popularity within the software engineering community, and sufficient labeled code smell instances. While some within-projects may no longer be actively developed, they provide a comprehensive representation of

TABLE I
WITHIN-PROJECTS USED IN THE STUDY

Projects	Releases	#Classes	LOC (K)	Description
GanttProject	V 1.10.2	245	41	A cross-platform project scheduling and management tool
ArgoUML	V 0.19.8	200	300	An open-source UML modeling tool for software development
Xerces-J	V 2.7.0	991	240	A Java-based XML parser for validating and parsing XML documents
JFreeChart	V 1.0.9	521	170	A Java library for creating charts, graphs, and other visualizations
Ant-Apache	V 1.7.0	1,839	327	A Java library for automating software builds and deployments
Azureus	V 2.3.0.6	1,449	42	A free, open-source peer-to-peer file sharing client for the BitTorrent protocol

TABLE II
CROSS-PROJECTS USED IN THE STUDY

Projects	Releases	#Classes	LOC (K)	Description
Hibernate	V 5.2.17	2,642	300	An ORM tool for Java
JMeter	V 3.3	1,092	96	A load testing tool for web applications
Tomcat	V 9.0.0.M22	3,172	68	An open-source web server and servlet container
POI	V 3.16	2,071	62	A Java library for working with Microsoft Office documents
Hadoop	V 3.1.1	4,030	367	A distributed computing platform

various code smell patterns and codebase characteristics. For cross-project analysis, we employed 5-fold cross-validation to ensure robust model evaluation. In the cross-project scenario, the RIs are passed to the optimized detectors that will be refined each time to get the best detectors.

B. Selection of the baseline approaches

Our experimental study involved assessing the performance of CLADES, which evolves DT detectors and adaptively updates the detectors by performing a clustering-based RIs selection. Additionally, we proposed a modified CLADES version that does not use SA in the adaptation module (this version was called CLADES_{wsa}) to assess the impact of removing SA on the performance of the detection process. Moreover, we selected baseline approaches including DECOR [2], GP [4], MOGP [12], and BLOP [13]. The rationale behind selecting these baseline approaches was twofold. First, it allowed us to gain a broader perspective on the performance of our proposed approach compared to existing methods, encompassing three categories of search-based methods: (1) Single-objective methods, (2) Multi-objective methods, and (3) Bilevel methods. Second, we included DECOR as a heuristic-based approach to understand if CLADES could outperform code smell detection methods relying on heuristics.

C. Training and configuration of our approach

1) *Training of the models:* In this study, our primary goal was to show the effectiveness of CLADES in detecting code smells and its potential to enhance the accuracy and efficiency of cross-project code smell detection. To achieve this, we initiated the process by extracting software metrics from selected projects using the Eclipse Metrics plugin tool, which includes

TABLE III
PARAMETERS CONFIGURATIONS

Parameters	CLADES	GP	MOGP	BLOP
Crossover rate	0.9	0.9	0.8	0.4
Mutation rate	0.1	0.1	0.2	0.7
Population size	200	100	100	30

several software metrics for each within-project software class (instance) within the dataset. Subsequently, we labeled these instances using five well-known tools, namely: DECOR [2], JDEODORANT¹, PMD², INFUSION³, and IPLASMA⁴. Remarkably, the instances were labeled automatically using these tools, eliminating the need for manual labeling. This process resulted in a dataset where each row represented a class from the project, while each column represented the value of a metric computed for that class.

2) *Parameters Tuning*: The proper tuning of an algorithm’s parameters significantly impacts its performance. In our work, we employed a trial-and-error approach to determine parameter values for our proposed GA and the compared approaches. For the other algorithms used in our comparison, we adopted the parameter values defined in their original papers (cf. Table III). To ensure a fair comparison, we kept a consistent stopping criterion for all algorithms. Specifically, we terminated the optimization process after 270,000 fitness evaluations. This criterion was chosen to accommodate all algorithms, including BLOP, which employs two populations. Both the upper-level and the lower-level populations are composed of 30 individuals and they undergo 15 generations and 20 generations, respectively. For the SA parameters, we set the T_{max} value to 1000, while the α reduction parameter was set to 0.1.

D. Performance Evaluation and Statistical Analysis

In our study, we evaluated the performance of our proposed approach and compared it with respect to existing algorithms using two quality metrics: F_β and AUC-PR [14]. The weighted F1-score served as the fitness function during the optimization process, while the AUC-PR provided an additional metric to comprehensively assess model performance. The AUC-PR metric measures the area under the precision-recall curve, offering robust performance evaluation, especially for imbalanced datasets. Its formula is given by:

$$AUC - PR = \sum_{i=1}^{n-1} (R_{i+1} - R_i) * P_{i+1} \quad (3)$$

where n is the number of points on the precision-recall curve, R_i is the recall, and P_i is the precision at the i^{th} threshold value, computed from DT classifications. The summation is taken over all threshold values where recall changes. The

AUC-PR ranges from 0 to 1, with higher values indicating better performance. To assess the significance of the results, we used the Wilcoxon test in a pairwise fashion with a p-value of 5%. The test results of the different comparisons are represented by (+) (significance) and (-) (no significance). The (+) symbol indicates that H_0 is rejected while the (-) symbol means that H_0 is accepted, where H_0 states that median (Algorithm 1) = median (Algorithm 2) and H_1 is median (Algorithm 1) \neq median (Algorithm 2). Each experiment was performed 31 times, and the comparisons between the algorithms were based on the median metrics’ values collected from the different approaches. The effect size measure employed was the Vargha-Delaney A measure, which indicates the magnitude of the performance difference between the algorithms. The Vargha-Delaney A measure was interpreted as follows: “large” if A is less than 0.29 or greater than 0.71, “medium” if A is less than 0.36 or greater than 0.64, and “small” if A is less than 0.64 or greater than 0.64.

V. RESULTS

The objective of this section is to present and clarify the comparative findings of our experimental evaluation. Our aim is to address the two research questions previously stated and to provide evidence supporting the effectiveness of key aspects of our CLADES approach. These aspects include its ability to: (1) adapt to varying project environments through the use of SA; (2) produce effective code smell detectors via a hybrid approach, and (3) achieve precise and scalable detection while minimizing the need for manual review. We will present results from both within-project and cross-project scenarios, comparing CLADES’ performance against various existing techniques.

A. RQ1 results

Table IV displays the results of the five detection methods that are trained on the within-projects and evaluated on the five cross-projects (cf. Tables I and II) with different sizes. Our proposed approach has the ability to treat the problem of imbalanced data, which mainly appears at the level of within projects and cross-projects. Based on the results presented in the table, it can be observed that CLADES performs better than all the other approaches with an F_β ranging between 55.20 and 46.13. Interestingly, CLADES_{wsa} shows competitive results, with an F_β ranging between 55.20 and 46.13. Such results indicate that the global GA search contributes significantly to code smell detection, but it cannot surpass the CLADES results. This could be explained by the fact that CLADES uses SA to perform a local and iterative refinement of the DTs making them able to detect the best search directions. BLOP had the second-best performance with an F_β score ranging from 38.59 to 41.62. On the other hand, DECOR, GP, and MOGP show comparatively poor results, with the highest F_β scores being 0.146, 0.161, and 0.216, respectively. The superior performance of CLADES compared to the remaining search-based approaches (i.e., GP, MOGP, and BLOP) can be attributed to its ability to consider the imbalance factor in

¹<https://github.com/tsantalidis/JDeodorant>

²<https://pmd.github.io/>

³<http://www.intooitus.com/products/infusion>

⁴<http://loose.upt.ro/iplasma/>

TABLE IV

SHOWS THE MEDIAN F_β AND AUC-PR SCORES OF CLADES, CLADES_{wsa}, DECOR, GP, MOGP, AND BLOP OVER 31 RUNS OF THE DETECTION TASK FOR EIGHT DIFFERENT SMELL TYPES TRAINED ON WITHIN PROJECTS. THE BEST AND SECOND-BEST F_β (OR AUC-PR) VALUES ARE IN BOLDFACE AND UNDERLINED, RESPECTIVELY.

	CLADES		CLADES _{wsa}		BLOP		MOGP		GP		DECOR	
	F_β %	AUC-PR	F_β %	AUC-PR	F_β %	AUC-PR	F_β %	AUC-PR	F_β %	AUC-PR	F_β %	AUC-PR
JMeter	54.36	0.5571	<u>50.44</u>	<u>0.5251</u>	40.52	0.4117	32.44	0.3319	28.02	0.2883	20.51	0.2097
	(+ + + + +)	(+ + + + +)	(+ + + + +)	(+ + + + +)	(+ + +)	(+ + +)	(+ +)	(+ +)	(-)	(-)		
Hibernate	55.20	0.5608	<u>51.13</u>	<u>0.5292</u>	41.62	0.4282	33.08	0.3389	26.65	0.2604	20.13	0.2064
	(+ + + + +)	(+ + + + +)	(+ + + + +)	(+ + + + +)	(+ + +)	(+ + +)	(+ +)	(+ +)	(-)	(-)		
POI	50.85	0.5135	<u>47.11</u>	<u>0.4875</u>	40.17	0.4157	31.70	0.3150	23.11	0.2432	18.72	0.1891
	(+ + + + +)	(+ + + + +)	(+ + + + +)	(+ + + + +)	(+ + +)	(+ + +)	(+ +)	(+ +)	(-)	(-)		
Tomcat	48.71	0.4989	<u>44.10</u>	<u>0.4530</u>	38.96	0.3905	30.63	0.3139	23.87	0.2409	13.45	0.1410
	(+ + + + +)	(+ + + + +)	(+ + + + +)	(+ + + + +)	(+ + +)	(+ + +)	(+ +)	(+ +)	(+)	(+)		
Hadoop	46.13	0.4796	<u>41.89</u>	<u>0.4357</u>	38.59	0.3887	29.11	0.3021	21.52	0.2214	12.93	0.1312
	(+ + + + +)	(+ + + + +)	(+ + + + +)	(+ + + + +)	(+ + +)	(+ + +)	(+ +)	(+ +)	(+)	(+)		

- A "+" sign at the i^{th} position indicates that the algorithm's F_β (or AUC-PR) median value is statistically different from the i th algorithm value.

- Conversely, a "-" sign indicates the opposite. Additionally, the table provides the effect sizes values (small (s), medium (m), and large (l)) using the Cohen's statistics.

the solution evaluation process, which is not accounted for in the other search-based approaches. Additionally, the rule-based (more precisely, DECOR) approach showed the poorest results with an F_β ranging between 12.93 and 20.51. This can be interpreted by the fact that the rules are designed manually and do not take into account the imbalance factor in the evaluation process, unlike the remaining compared approaches including CLADES.

The results of the AUC-PR metric are consistent with those of the F_β . CLADES achieves the highest AUC-PR values, followed by CLADES_{wsa}. This means that both methods perform better than existing approaches when it comes to handling imbalanced data. CLADES outperforms its version without SA because the use of SA allows it to locally refine the model based on the particular data characteristics. The adoption of SA helps CLADES to improve its robustness and generalization capability. However, existing approaches such as DECOR, GP, MOGP, and BLOP suffer from bias and are highly susceptible to stochastic noise due to their inability to handle imbalanced data. DECOR exhibits the poorest performance in the presence of data imbalance, primarily because the manually defined detection rules are unable to capture all possible smells present in imbalanced data. Although certain rules may identify some smells, they cannot capture them all, leading to subpar performance. Generating a large number of rules manually to detect all types of smells in imbalanced data is still a challenging task. The results shown in Table IV indicate that all the detection methods, including CLADES, exhibit medium to poor performance when trained on within-projects and evaluated on unlabelled cross-projects. One of the main reasons for these results can be attributed to the differences in data distributions between the two types of projects. In within-projects, the distribution of code smells may differ from cross-projects due to variations in coding styles, project scopes, and application domains. This can lead to a mismatch between the training and testing data, resulting in a poorer performance. Additionally, cross-projects evaluation can be more challenging due to the lack of labeled

data and differences in project structure and complexity.

B. RQ2 results

Table V presents the results of several approaches, including CLADES, which are trained on within-projects and updated using RIs selected through the K-means approach from cross-projects. These approaches are then evaluated on the remaining unlabelled instances of the cross-projects. It is important to note that the selected instances are labeled by advisors and supervised by expert developers. CLADES leverages an incremental approach that fine-tunes its DT detectors iteratively to adapt to the evolving characteristics of each new project. The obtained results show that our proposed approach outperforms the remaining approaches, with F_β values ranging from 88.54 to 94.09 and AUC-PR values ranging from 0.9013 to 0.947. Among the remaining approaches, the BLOP approach achieved the best F_β and AUC-PR scores of 45.75 and 0.4652, respectively. However, the rule-based approach (DECOR) produced the worst results, with the best F_β and AUC-PR scores of 23.74 and 0.2418, respectively. These results highlight the benefit of CLADES' hybrid approach, which combines a global GA search with a local SA refinement, both informed by a clustering-based RI selection. Furthermore, they highlight that the SA component contributes significantly to the adaptability of CLADES in cross-project scenarios, as CLADES_{wsa} shows lower performance compared to CLADES. Specifically, while both CLADES and CLADES_{wsa} leverage the GA to identify a good initial solution based on the within-project data, only CLADES employs the SA algorithm to locally refine the solution to the characteristics of the new cross-project data based on the RIs. This allows it to perform better due to its greater adaptability. The results of the remaining approaches (GP, MOGP, and BLOP) can be explained by the fact that they are often trained on initial data that may no longer be fully representative of the new data, which leads to poor generalization when the data distributions are different. The poor performance of the DECOR approach in cross-project evaluation can be attributed to the manually fixed

TABLE V

THE MEDIAN F_β AND AUC-PR SCORES OF CLADES, CLADES_{wsa}, DECOR, GP, MOGP, AND BLOP OVER 31 RUNS OF THE DETECTION TASK FOR EIGHT DIFFERENT SMELL TYPES TRAINED ON WITHIN PROJECTS AND SELECTED INSTANCES FROM CROSS-PROJECTS. THE BEST AND SECOND-BEST F_β (OR AUC-PR) VALUES ARE IN BOLDFACE AND UNDERLINED, RESPECTIVELY.

	CLADES		CLADES _{wsa}		BLOP		MOGP		GP		DECOR	
	F_β %	AUC-PR	F_β %	AUC-PR	F_β %	AUC-PR	F_β %	AUC-PR	F_β %	AUC-PR	F_β %	AUC-PR
JMeter	93.21	0.9417	<u>74.43</u>	<u>0.7765</u>	45.75	0.4652	34.21	0.3520	31.10	0.3287	23.74	0.2418
	(+ + + + +)	(+ + + + +)	(+ + + +)	(+ + + +)	(+ + +)	(+ + +)	(- +)	(- +)	(+)	(+)		
	(l l l l l)	(l l l l l)	(l l l l)	(l l l l)	(m l l)	(m l l)	(s m)	(s m)	(m)	(m)		
Hibernate	94.09	0.9473	<u>74.76</u>	<u>0.7745</u>	43.33	0.4404	33.98	0.3553	32.47	0.3206	22.19	0.2363
	(+ + + + +)	(+ + + + +)	(+ + + +)	(+ + + +)	(+ + +)	(+ + +)	(- +)	(- +)	(+)	(+)		
	(l l l l l)	(l l l l l)	(l l l l)	(l l l l)	(m l l)	(m l l)	(s m)	(s m)	(m)	(m)		
POI	93.20	0.9405	<u>68.36</u>	<u>0.7034</u>	41.52	0.4358	32.51	0.3311	28.79	0.2917	20.40	0.2130
	(+ + + + +)	(+ + + + +)	(+ + + +)	(+ + + +)	(+ + +)	(+ + +)	(- +)	(- +)	(+)	(+)		
	(l l l l l)	(l l l l l)	(l l l l)	(l l l l)	(m l l)	(m l l)	(s m)	(s m)	(m)	(m)		
Tomcat	89.87	0.9192	<u>60.45</u>	<u>0.6096</u>	40.20	0.4193	31.39	0.3210	24.68	0.2487	15.52	0.1606
	(+ + + + +)	(+ + + + +)	(+ + + +)	(+ + + +)	(+ + +)	(+ + +)	(+ +)	(+ +)	(+)	(+)		
	(l l l l l)	(l l l l l)	(l l l l)	(l l l l)	(m l l)	(m l l)	(m m)	(m m)	(m)	(m)		
Hadoop	88.54	0.9013	<u>68.20</u>	<u>0.6857</u>	39.87	0.4019	30.16	0.3096	23.14	0.2463	13.10	0.1427
	(+ + + + +)	(+ + + + +)	(+ + + +)	(+ + + +)	(+ + +)	(+ + +)	(+ +)	(+ +)	(+)	(+)		
	(l l l l l)	(l l l l l)	(l l l l)	(l l l l)	(m l l)	(m l l)	(m m)	(m m)	(m)	(m)		

- A “+” sign at the i^{th} position indicates that the algorithm’s F_β (or AUC-PR) median value is statistically different from the i^{th} algorithm’s value.

- Conversely, a “-” sign indicates the opposite. Additionally, the table provides the effect sizes values (small (s), medium (m), and large (l)) using Cohen’s statistics.

detection rules used in DECOR, which may not be able to effectively capture all potential code smells present in cross-projects due to the different data distributions. In contrast, CLADES utilizes a clustering technique for selecting RIs from new projects, ensuring that it is exposed to a diverse set of smell patterns. These RIs are then added to the training data, enabling incremental adaptation. The SA algorithm then fine tunes the structure of decision trees by modifying split points, feature weights, and sub-tree structures, allowing for continuous adaptation with respect to the new project. Finally, the local refinement process allows CLADES to explore the search space effectively and fine-tune its DT detectors to capture the unique characteristics of each new project. This adaptability is crucial for achieving high performance across different projects. For example, when evaluating CLADES on the Hibernate project, the model successfully adapted to specific code smells such as ‘God Class’ and ‘Long Method’. Our experimental results, including the consistent performance across different projects as shown in Table II, validate the generalization capability of our approach.

VI. RELATED WORK

The detection of code smells is still a critical research topic in the field of software engineering. Numerous approaches have been proposed for the automatic detection of code smells. These methods can help software engineers to perform the detection task or to design the detectors. Typically, the proposed methods can be classified into three categories: (1) Rule-based approaches, (2) Machine learning-based approaches, and (3) Search-based ones. Rule-based approaches have been used to identify code smells by combining several metrics, with each metric being subjected to threshold calibration by the software engineer. However, setting the threshold values accurately is challenging and can result in different definitions of code smells and low accuracy detection. An example of a rule-based approach is DECOR [2], which uses predefined rule-cards to identify symptoms of code smells based on code metrics. While effective for simple code smells, they often

struggle with complex smells like Feature Envy and God Class, which require analyzing deeper relationships within the code. Machine learning-based approaches have been proposed to overcome the limitations of rule-based approaches. These methods train classifiers with datasets to make them able to predict the label of code smell instances. Several classifier models have been used, including DTs [3], Naïve Bayes [15], support vector machines [16], and deep neural networks [17]. Search-based approaches use metaheuristics to optimize a set of detectors to improve the detection accuracy. Among them, GAs are commonly used due to their ability to avoid local optima and their probabilistic acceptance of fitness degradation. Examples of search-based approaches include GP [4], MOGP [13], and BLOP [12].

Recent advancements have explored the integration of local search techniques within GAs to enhance their adaptability and performance in dynamic contexts. For instance, hybrid approaches combining multi-parent evolution with differential evolution [18] have shown improved efficiency and robustness in global optimization. Additionally, memetic algorithms incorporating breakout local search [19] have been effective in solving domain-specific problems like the generalized traveling salesman problem. These approaches have shown their ability to refine solutions and adapt to problem-specific constraints [20]. These hybrid methods highlight the potential of combining global search strategies with local refinements to address the complexities of dynamic optimization problems more effectively.

Few machine learning-based approaches have been proposed to deal with cross-project smell detection. The study presented in [21] proposes a novel cross-project machine learning approach to address the imbalanced datasets problem in code smell detection. By utilizing transfer learning, the authors increase the number of smelly instances in training datasets and compare the performance of cross-project and within-project classification approaches. The results show that the cross-project classification approach performs similarly to the within-project approach. Despite the recent progress

in cross-project code smell detection, there are still some challenges that need to be addressed. One challenge is that the performance of cross-project code smell detection approaches can be degraded by the domain gap between the training dataset and the target dataset. Another challenge is that the training of cross-project code smell detection models can be computationally expensive. Our approach is motivated by the need to develop a cross-project code smell detection approach that is robust to the domain gap and computationally efficient. This combination allows our approach to learn from a small amount of data and to adapt to the target dataset.

VII. CONCLUSION AND FUTURE WORK

In this paper, we introduced CLADES, an evolutionary approach for cross-project code smell detection. Our proposed approach evolves a set of DT detectors using a GA. The main characteristic of CLADES is that it transfers the optimization knowledge from within-project detectors to cross-project ones. Therefore, once a new project is received, it performs a clustering-based RIs selection and integrates the information provided by the RIs into the current generated detectors using SA. This local search approach is applied to rectify the search directions and to guide the population towards the new optimal cross-project detectors. Experiments on five open-source projects showed that CLADES outperforms several baseline methods in terms of both accuracy and efficiency, providing a robust tool for identifying code smells in diverse software systems.

As part of our future work, we plan to explore several directions to further improve the effectiveness and efficiency of CLADES. First, we plan to investigate the use of other incremental learning algorithms, such as Online Artificial Neural Networks [22], to further enhance the incremental learning process. Second, we aim to employ ensemble methods, such as bagging or boosting, to further enhance the detection accuracy of our approach [23]. Third, it will be interesting to further investigate the use of other clustering techniques, such as spectral clustering [24] and density-based clustering [25] for the clustering-based instance selection process. Finally, we plan to apply CLADES to more diverse and larger software systems to evaluate their scalability and generalization capability.

REFERENCES

- [1] G. Santos, A. Santana, G. Vale, and E. Figueiredo, "Yet another model! a study on model's similarities for defect and code smells," in *Fundamental Approaches to Software Engineering: 26th International Conference, FASE 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22–27, 2023, Proceedings*. Springer, 2023, pp. 282–305.
- [2] N. Moha, Y. G. Gueheneuc, L. Duchien, and A. F. L. Meur, "DECOR: A Method for the Specification and Detection of Code and Design Smells," *IEEE Transactions on Software Engineering*, vol. 36, no. 1, pp. 20–36, 2010.
- [3] L. Amorim, E. Costa, N. Antunes, B. Fonseca, and M. Ribeiro, "Experience report: Evaluating the effectiveness of decision trees for detecting code smells," in *Proceedings of the 26th International Symposium on Software Reliability Engineering*. IEEE, 2015, pp. 261–269.
- [4] A. Ouni, M. Kessentini, H. Sahraoui, and M. Boukadoum, "Maintainability defects detection and correction: a multi-objective approach," *Automated Software Engineering*, vol. 20, no. 1, pp. 47–79, 2013.
- [5] A. Ho, A. M. Bui, P. T. Nguyen, and A. Di Salle, "Fusion of deep convolutional and lstm recurrent neural networks for automated detection of code smells," in *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*, 2023, pp. 229–234.
- [6] R. Gupta and S. K. Singh, "Using machine learning for inter-smell detection: A feasibility study," in *Artificial Intelligence and Data Science: First International Conference, ICAIDS 2021, Hyderabad, India, December 17–18, 2021, Revised Selected Papers*. Springer, 2022, pp. 291–305.
- [7] M. A. Al Hilmi, A. Puspaningrum, D. O. Siahaan, H. S. Samosir, A. S. Rahma *et al.*, "Research trends, detection methods, practices, and challenges in code smell: Slr," *IEEE Access*, vol. 11, pp. 129 536–129 551, 2023.
- [8] F. Zhang, Z. Zhang, J. W. Keung, X. Tang, Z. Yang, X. Yu, and W. Hu, "Data preparation for deep learning based code smell detection: A systematic literature review," *Journal of Systems and Software*, vol. 209, pp. 1–18, 2024.
- [9] J. R. Quinlan, *C4. 5: programs for machine learning*. Elsevier, 2014.
- [10] M. A. Arostegui Jr, S. N. Kadipasaoglu, and B. M. Khumawala, "An empirical comparison of tabu search, simulated annealing, and genetic algorithms for facilities location problems," *International Journal of Production Economics*, vol. 103, no. 2, pp. 742–754, 2006.
- [11] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2009, vol. 2.
- [12] D. Sahin, M. Kessentini, S. Bechikh, and K. Deb, "Code-Smell Detection as a Bilevel Problem," *ACM Transactions on Software Engineering and Methodology*, vol. 24, no. 1, pp. 1–44, 2014.
- [13] U. Mansoor, M. Kessentini, B. R. Maxim, and K. Deb, "Multi-objective code-smells detection using good and bad design examples," *Software Quality Journal*, vol. 25, no. 2, pp. 529–552, 2017.
- [14] S. A. Khan and Z. A. Rana, "Evaluating performance of software defect prediction models using area under precision-recall curve (auc-pr)," in *2019 2nd International Conference on Advancements in Computational Sciences (ICACS)*. IEEE, 2019, pp. 1–6.
- [15] F. Khomh, S. Vaucher, Y.-G. Guéhéneuc, and H. Sahraoui, "A bayesian approach for the detection of code and design smells," in *Proceedings of the 9th International Conference on Quality Software*. IEEE, 2009, pp. 305–314.
- [16] A. Maiga, N. Ali, N. Bhattacharya, A. Sabane, Y.-G. Gueheneuc, and E. Aimeur, "SMURF: A SVM-based incremental anti-pattern detection approach," in *Proceedings of the 19th Working conference on Reverse engineering*. IEEE, 2012, pp. 466–475.
- [17] M. White, M. Tufano, C. Vendome, and D. Poshvanyk, "Deep learning code fragments for code clone detection," in *Proceedings of the 31st IEEE/ACM international conference on automated software engineering*, 2016, pp. 87–98.
- [18] W. Zhang and Y. Lan, "A novel memetic algorithm based on multiparent evolution and adaptive local search for large-scale global optimization," *Computational Intelligence and Neuroscience*, vol. 2022, no. 1, p. 3558385, 2022.
- [19] M. El Krari, B. Ahiod, and B. El Benani, "A memetic algorithm based on breakout local search for the generalized traveling salesman problem," *Applied Artificial Intelligence*, vol. 34, no. 7, pp. 537–549, 2020.
- [20] S. Bechikh, "Incorporating decision maker's preference information in evolutionary multi-objective optimization," Ph.D. dissertation, High Institute of Management of Tunis, University of Tunis, Tunisia, 2013.
- [21] M. De Stefano, F. Pecorelli, F. Palomba, and A. De Lucia, "Comparing within- and cross-project machine learning algorithms for code smell detection," in *Proceedings of the 5th International Workshop on Machine Learning Techniques for Software Quality Evolution*, 2021, pp. 1–6.
- [22] D. Q. Zhou, U. D. Annakkage, and A. D. Rajapakse, "Online monitoring of voltage stability margin using an artificial neural network," *IEEE Transactions on Power Systems*, vol. 25, no. 3, pp. 1566–1574, 2010.
- [23] T. G. Dietterich, "Ensemble methods in machine learning," in *Multiple Classifier Systems: First International Workshop, MCS 2000 Cagliari, Italy, June 21–23, 2000 Proceedings 1*. Springer, 2000, pp. 1–15.
- [24] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, pp. 395–416, 2007.
- [25] H.-P. Kriegel, P. Kröger, J. Sander, and A. Zimek, "Density-based clustering," *Wiley interdisciplinary reviews: data mining and knowledge discovery*, vol. 1, no. 3, pp. 231–240, 2011.