A Possibilistic Evolutionary Approach to Handle the Uncertainty of Software Metrics Thresholds in Code Smells Detection

Sofien Boutaib[1], Maha Elarbi[1], Slim Bechikh[1], Fabio Palomba[2], Lamjed Ben Said[1]
[1]SMART lab, University of Tunis, ISG, Tunisia
[2]SeSa Lab - University of Salerno, Italy
mohamedsofien.boutaib@isg.u-tunis.tn

***Abstract*—A code smells detection rule is a combination of metrics with their corresponding crisp thresholds and labels. The goal of this paper is to deal with metrics' thresholds uncertainty; as usually such thresholds could not be exactly determined to judge the smelliness of a particular software class. To deal with this issue, we first propose to encode each metric value into a binary possibility distribution with respect to a threshold computed from a discretization technique; using the Possibilistic C-means classifier. Then, we propose ADIPOK-UMT as an evolutionary algorithm that evolves a population of PK-NN classifiers for the detection of smells under thresholds' uncertainty. The experimental results reveal that the possibility distribution-based encoding allows the implicit weighting of software metrics (features) with respect to their computed discretization thresholds. Moreover, ADIPOK-UMT is shown to outperform four relevant state-of-art approaches on a set of commonly adopted benchmark software systems.**

***Index Terms*—Code smells detection, uncertain metrics' thresholds, possibility distribution, possibilistic K-NN, evolutionary algorithm**

I. Introduction

The problem of code smell detection, i.e., the detection of source code quality issues, is still relevant in the software engineering research community, despite the effort spent in the last decades [1]–[3]. In fact, code smells (anti-patterns) [4] are associated to poor design or implementation of the source code that hamper the software maintenance process and may contribute to prone the occurrence of bugs during time [5]. A considerable number of detection methods have been suggested in the literature in which the used detectors principally differ in the underlying technique such as rule-based techniques, and machine learning-based ones. Despite the good performance presented by these detectors, the search-based detectors have shown their superiority comparing to the other approaches thanks to their global search capability. Code smell detection problem could be considered mainly as a classification one. Therefore, the software classes are seen as records, the structural software metrics are seen as features, and the existence of code smell or not coincide with the class labels. Similarly, to the detection problem, the identification problem with a small modification consisting in assigning a smell type to a class label. In this way, detection could be formed as a binary classification problem, while identification is a multi-class one.

Software engineers could have conflictual opinions about the metrics thresholds of the detection rules as well as the smell type identification ones [6]. As the data classification problem, the code smell detection problem could have uncertain metrics' thresholds. For example, the specification of the metric threshold value related to the long parameter list smell type differs from one engineer to another. This is essentially due to the subjective nature of human being reflection, its knowledge, and its experience. Unfortunately, the existing works in the literature, including SBSE (Search-Based Software Engineering) ones, do not deal with the uncertainty issue. In this context, rejecting and/or ignoring uncertain data can cause a loss of information and significantly deteriorates the detection performance [7]. To deal with the uncertainty aspect, various uncertainty tools have been proposed such as Fuzzy set theory [8], Rough set theory [9], Belief Function Theory [10], and Possibility theory [8]. Motivated by the interesting performance of the PK-NN (Possibilistic K-NN) [11] algorithm in classifying uncertain data. In this work, we aim to propose a new search-based method, named ADIPOK-UMT (Anti-patterns Detection and Identification using Possibilistic Optimized K-NNs under Uncertain Metrics' Thresholds) that evolves a set of smells detectors (each detector is a PK-NN) using an evolutionary algorithm, more precisely, the Genetic Algorithm (GA). To sum up, the principal contributions of this paper are:

1) For the case of detection, a Base of Examples (BE) that includes software classes characterized by uncertain metrics' thresholds values and certain class labels is developed, where class label could be "smelly" and/or "non-smelly" while each instance (i.e., software class) pertains to the values of the considered metrics' thresholds in the BE with likelihood degrees;
2) For the case of identification, a BE of smelly classes characterized by uncertain metrics' thresholds values and certain class labels is designed, where the class labels are crisp and the software classes are assigned to each value of each quality metric threshold value with

different likelihood degrees;
3) Proposing ADIPOK-UMT as a new SBSE method for smells detection and identification under uncertainty;
4) Showing the outperformance of ADIPOK-UMT comparing to the most prominent baseline methods in the literature on six open source software systems.

The rest of this paper is organized as follows. Section II briefly presents the necessary background of the possibility theory. The ADIPOK-UMT approch that allows coping with uncertain metrics' thresholds values, which mimics the subjectivity of the software engineer on Section III. Section IV describes the experimental setup, while Section V presents and analyses the comparative results. Then, Section VI summarizes the related work. Finally, Section VII provides conclusion and presents some future avenues.

II. Possibility theory: Basic concepts

In order to handle the uncertainty aspect, we propose to use the possibility theory. In this section, we aim to show the fundamental concepts of this theory used over this paper.

The Possibility theory is used to model the imprecise and uncertain information. It is based on the possibility distribution function denoted by π, which allows to associate each hypothesis ω_i of the universe of discourse $\Omega = \omega_1, ..., \omega_n$ a value called possibility degree in [0, 1].

By convention, for whatever hypothesis $\omega_i \in \Omega, \pi(\omega_i) = 1$, signifies that ω_i is completely possible and $\pi(\omega_i) = 0$, means that the realization of ω_i is an impossible hypothesis (cannot be the real word). A possibility distribution is considered normalized if there is at least one hypothesis $\omega_i \in \Omega$, where $max\ \omega \in \Omega, \{\pi(\omega)\} = 1$. Over this paper, only the normalized possibility distributions are used.

To measure the similarity between two possibility distributions (π_1 and π_2), we need to utilize a similarity measure able to deal with the uncertain attribute value. To this end, we propose to adopt the possibilistic similarity measure called *Information Affinity measure* (see Equation 1) (denoted by Affinity) [12].

$$Affinity\ (\pi_1, \pi_2) = 1 - \frac{D(\pi_1, \pi_2) + Inc(\pi_1, \pi_2)}{2} \quad (1)$$

where $D(\pi_1, \pi_2)$[1] corresponds to normalized Manhatten distance. $Inc(\pi_1, \pi_2)$ is the inconsistency and it is useful in evaluating the conflictual degree among two possibility distributions π_1 and π_2 which is presented by $Inc(\pi_1, \pi_2)$ [2].

III. Proposed approach: ADIPOK-UMT

This section presents our proposed ADIPOK-UMT method. First, we present an overview of the detectors generation process working principle. Then, we describe how the BE with uncertain metrics' thresholds values are constructed. Besides, we describe the details of the GA operators including the adaptation of an adequate fitness assignment, the selection, and the variation operators. Finally, we show how the software engineer could employ the obtained PK-NN detectors for the detection and/or identification of smell types.

[1] $D(\pi_1, \pi_2) = \frac{1}{n} \times \sum_{i=1}^{n} |\pi_1(\omega_i) - \pi_1(\omega_i)|$; $\forall \omega_i \in \Omega = \omega_1, ..., \omega_n$ and $n = |\Omega|$

[2] $Inc(\pi_1, \pi_2) = 1 - max_\omega(\pi_1(\omega), \pi_2(\omega))$ with $\wedge$ is selected as the minimum operator.

TABLE I
List of code smells considered in this work.

Code Smell / Antipattern	Description
God Class (aka Blob)	It occurs when a class centralizes an important part of the system behaviour while the remaining classes mainly contain data.
Data Class	It happens in the case where a class retains only data and not the complex functionalities.
Feature Envy	It arises when a method invokes much more methods from an external class than the invoked methods from the same englobing class.
Long Method	This smell refers to a method that is broad in terms of lines of code.
Duplicate code	Such smell arises when there are many redundancies for the same code fragment in many classes.
Long Parameter List	This smell happens when a method has a high number of parameter list.
Spaghetti Code	Such smell appears when the code structure is becoming much more complex and tangled.
Functional Decomposition	It arises when a class is developed in order to carry out a single function. This smell type is caused by the lack of OO developers experiences.

A. A Brief Overview of ADIPOK-UMT

As shown by Figure 1, the first step to develop our ADIPOK-UMT approach is the construction of the Possibilistic BE (PBE). The first step to develop our ADIPOK-UMT approach is the construction of the Possibilistic BE (PBE). In this study, we considered six open-source software projects. Based on the experimental studies of the existing works, we have used 22 structural metrics (cf. Table II). Our proposed approach could work independently from the existing code smell types. To investigate the performance of our ADIPOK-UMT, we have chosen 8 code smell types (cf. Table I). As the BE includes continuous metrics' values, a discretization technique (called Random Boundary Threshold (RBT)) is applied to convert these continuous values into categorical ones (i.e., thresholds) and then the Possibilistic C-Means classifier [13] is employed to generate the possibility distribution for each metric threshold value. Such process is used to mimic the uncertainty of the expert regarding the metrics thresholds values. Moreover, these binary possibility distributions (having the form of [1, X] or [X, 1]) in addition to the fact they present the uncertainty of the expert, they serve later to indicate the importance of each metric with its corresponding threshold value in the detection rule. Roughly speaking, once the PBE is created (cf. Figure 2), the GA evolves a number of PK-NN classifiers by optimizing a fitness function (i.e., *F-measure*) that is calculated according to the BE. Once a number of optimized possibilistic detectors are created, they are utilized to detect and/or identify the existing anti-patterns on unseen software classes (not used in the training phase) (cf. Figure 1). We remind that for the detection task, the PBE includes both smelly and non-smelly classes; while for the identification task, the PBE includes a single smell type.

B. Creation of Artificial Possibilistic BE

The real-world environment of the SE industry is far from being ideal. Similar to other fields, the uncertainty problem also exists in SE. Indeed, the BE could be submerged by

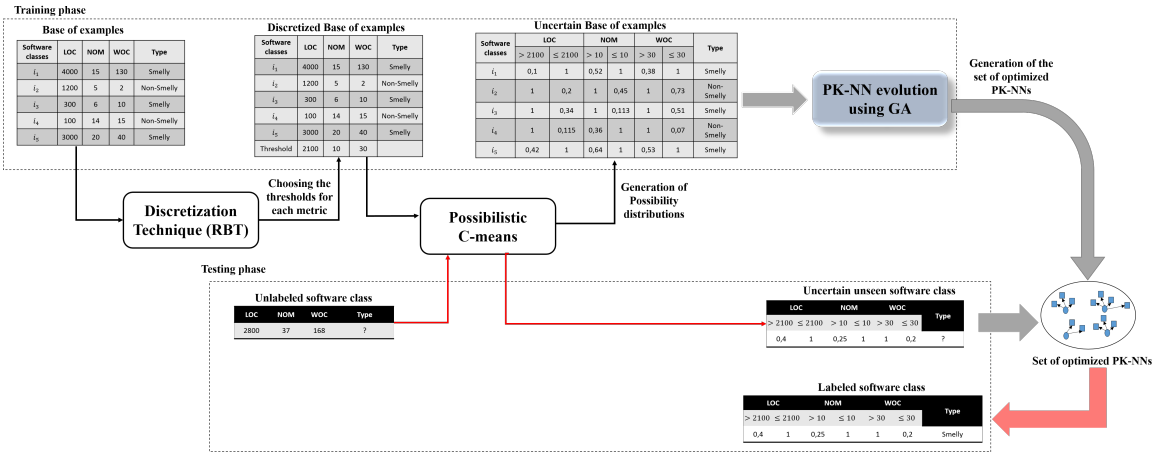

Fig. 1. A step by step illustration of the main framework of ADIPOK-UMT.

TABLE II
LIST OF QUALITY METRICS USED IN THIS WORK.

Metric	Description
ANA	Average Number of Ancestors.
AOFD	Access Of Foreign Data.
CAM	Cohesion Among Methods of Class.
CBO	Coupling Between Objects.
CIS	Class Interface Size.
CM	Changing Method.
DAM	Data Access Metric.
DCC	Direct Class Coupling.
DSC	Design Size in Classes.
LOC	Lines of Code.
MFA	Measure of Functional Abstraction.
MOA	Measure of Aggregation.
NOA	Number of Attributes.
NOAM	Number of Accessor Method.
NOF	Number of Fields.
NOH	Number of Hierarchies.
NOM	Number of Methods.
NOPA	Number of Public Attributes.
NPA	Number of Private Attributes.
TCC	Tight Class Cohesion.
WMC	Weighted Methods per Class.
WOC	Weighted Of Class.

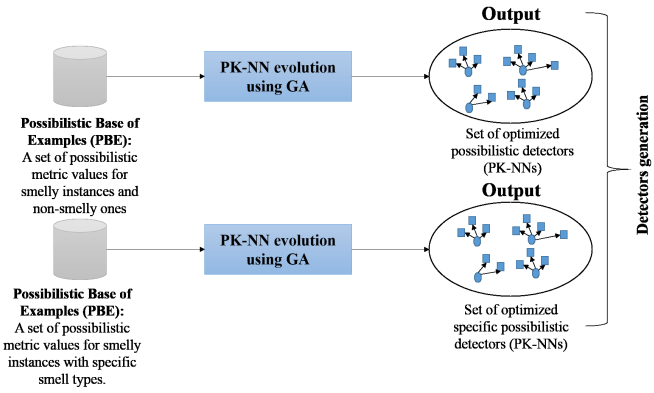

Fig. 2. The main schema of the ADIPOK-UMT approach.

subjectivity and doubtfulness of the software engineer at the level of metrics' values. This uncertainty could be explained by the fact that the software engineers may suffer from the lack of expertise or knowledge. Therefore, the software engineer is able to express its opinion in the form of likelihood values each corresponding to the membership of each attribute values for a given software class. To cope with such uncertainty, the possibility theory is a good solution since each software class could be assigned a set of metrics' thresholds values each having possibility degrees.

As the SE studies have not treat any metrics' thresholds uncertainty in the BE construction level, we have proposed in this work a construction process of a PBE, which could be performed as follows. First, the numerical BE is transformed into a categorical one using an existing discretization technique [14]. All the software classes in the BE are sorted in ascending order with respect to the values of the feature that will be discretized according to the RBT method. This latter works as follows. A boundary thresholds are identified as illustrated by Figure 3. In fact, a boundary threshold of a given feature corresponds to a midpoint among a successive pair of software classes (i.e., one of the instance is positive and the next one is negative). Finally, the threshold value of the considered feature (metric) is randomly chosen from the effective created thresholds as shown by Figure 3.

After transforming the BE into categorical one, the Possibilistic C-means (PCM) [13] is applied on a a discretised BE with certain metrics' thresholds values to come up with possibility distribution for each metric threshold of each instance (see Figure 4). PCM begins by creating the clusters for each metric threshold. According to one attribute, PCM estimates the distance among every instance and every cluster' center. A possibility distribution is created based on the calculated distance. Then, PCM attempts to minimize the objective function (see Equation 2). PCM calculates recursively the center of the clusters till the objective function stops changing. The PCM permits us to create for every instance, according to each BE metric threshold, a possibility distribution that corresponds to its membership to every cluster.

$$J_{PCM}(L,U) = \sum_{j=1}^{C}\sum_{i=1}^{N} \omega_{ij}^{\alpha} dist_{ij}^{2} + \sum_{j=1}^{C} \eta_j \sum_{i=1}^{N} (1-\omega_{ij})^{\alpha} \quad (2)$$

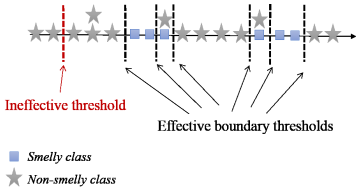

Fig. 3. The discretization technique.

$$u_{ij} = \frac{1}{1 + (\frac{d_{ij}^2}{\eta_i})^{\frac{1}{\alpha - 1}}} \tag{3}$$

where C is the number of clusters, N is the total number of metrics vectors in the BE, $dist_{ij}$ corresponds to the distance of a metric x_j and a prototype β_i, α represents a weighting exponent called the fuzzifier, which belongs to $[\ 1, \infty)$, η is the penalty parameter that is calculated based on Equation 3, L = β_1, ..., β_C is a C-tuple of prototypes, $U = [u_{ij}]$ is a $C \times N$ that corresponds to a matrix of possibilistic membership degrees. Here, w_{ij} is the membership degree of the metric x_j in cluster β_i. It is important to note that the idea of converting the attribute values into uncertain ones using PCM is inspired from the work of [15].

C. Individual encoding

The PK-NN performance depends on two parameters: (1) the FS (Feature selection), and (2) the number of nearest neighbors K. To optimize the parameters, the solution encoding in ADIPOK-UMT is a vector including all these parameters as depicted by Figure 5.

As these parameters correspond to different numerical types, we choose to apply the crossover per part as follows:

- The FS subset corresponds to a binary vector and is varied via the uniform crossover [16];
- The K value is an integer in [1, T] and is varied via the SBX crossover [16] with rounding to the nearest integer;

It is important to know that we randomly initialize these parameters at the beginning of the GA process, while representing the definition domain of every gene.

D. Fitness function assignment

This operator gives a quality value to every solution (PK-NN) of the GA's population. It is started after the population initialization process and whenever a new offspring PK-NN is born via the reproduction operators (crossover and mutation). The generated fitness value will be later employed over the selection process. It is worth noting that code smells detection is considered as a binary classification problem. This operator gives a quality value to every solution (PK-NN) of the GAs population. It is started after the population initialization process and whenever a new offspring PK-NN is born via the reproduction operators (crossover and mutation). The generated fitness value will be later employed over the selection process. It is worth noting that code smells detection is considered as a binary classification problem. The classification process of each PK-NN is performed as follows. First, we determine

TABLE III
CONFUSION MATRIX.

	Predicted class	
Actual class	Positive	Negative
Positive (Smelly)	TP	FN
Negative (Non-smelly)	FP	TN

the similarity between a given software class ($Class$) and a software class existing in the BE ($class_l$) using the affinity formula as follows:

$$affinity(class_l, Class) = \frac{\sum_{j=1}^{m} Affinity(\pi_{class_{lj}}, \pi_{Class_j})}{m} \tag{4}$$

where m represents the number of metrics (features), $\pi_{class_{lj}}$ represents the possibility distribution of a software class l existing in the BE with respect to a metric j, and π_{Class_j} represents the possibility distribution of an unseen software class ($Class$) with respect to the metric j, respectively. Second, we choose the K closest software classes to the unseen class based on the obtained similarity values. Finally, the majority voting is performed according to the proposed labels of the chosen instances in order to predict the label of the unseen software class. Thus, we use the confusion matrix (cf. Table III) to assess the performance of the detectors. Formally, TP (True Positives) is the number of actual smelly classes correctly classified, TN (True Negatives) is the number of actual non-smelly classes correctly classified, FN (False Negatives) is the number of actual non-smelly classes misclassified as smelly ones, and FP (False Positives) is the number of actual smelly classes misclassified as non-smelly ones. Based on this matrix, the widely used metric is the Accuracy. Despite the relevant information presented by this metric, we should not rely on it as it is biased due to the fact that the smelly and non-smelly classes have different sizes [17]. Such bias could be managed using the *F-measure* (see Equation 5) [18] metric as fitness function as it is known by its insensitivity to the imbalanced class sizes in binary classification.

$$\textit{F-measure} = \frac{2 * TP}{(2 * TP + FP + FN)} \tag{5}$$

E. Mating selection operator

The ADIPOK-UMT method encourages the selection of good parents in each generation of the GA and gives chance to the less effective individuals to participate in the reproduction process in the aim to avoid the local optima and approach the globally optimum detectors (PK-NN). Based on this fact, the binary tournament selection operator [19] is used to select parents for reproduction. In fact, in each generation (N/2) individuals are chosen as parents as follows. In every iteration of the parent selection process, two individuals are randomly selected and the best (the fittest) one is kept and added to the mating pool. This step is repeated with replacement till the fulfilment of the mating pool. As the two chosen individuals

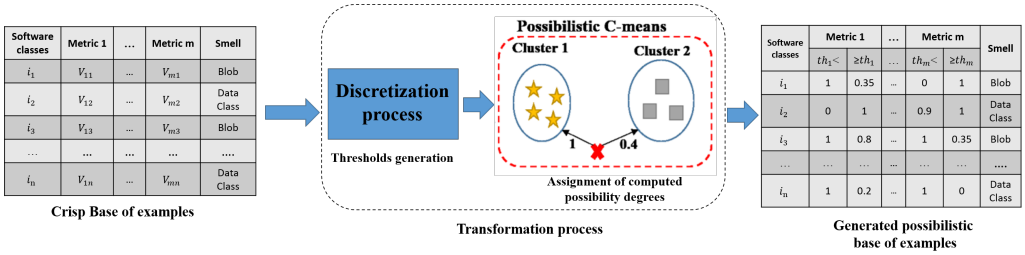

Fig. 4. Possibilistic BE generation process.

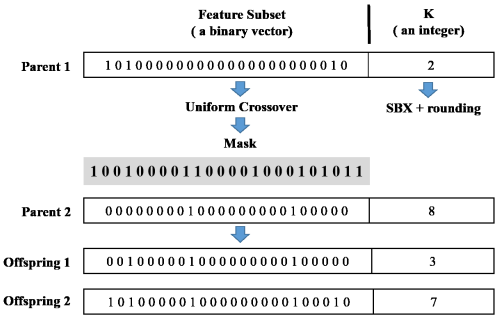

Fig. 5. Crossover opertaor of ADIPOK-UMT.

in every iteration is randomly carried out, the parents could have good or bad quality (fitness) values. This permits the acceptance of worse movements in the search space; which permits avoiding locally-optimal PK-NN models. It is worth noting that the convergence of the GA is assured by the choice of the best individual between the chosen parents.

F. Reproduction operators

As we mentioned previously (in Sub-section III-C), the chromosome is represented by two parts where the first one is represented by a binary string and the second one is an integer. As shown by Figure 5, the crossover is carried out per part. The binary string part is varied via the uniform crossover [16], which in turns utilizes a random binary mask. The K real value is varied via the SBX crossover operator with rounding [16]. As the SBX generates real numbers and the K value should be an integer value, the generated SBX value is rounded to the nearest integer.

The mutation operator is also applied per part. The one-point mutation operator [20] is applied on the first part, while the second part is mutated using the polynomial mutation operator for real numbers [16] with rounding to the nearest integer.

G. Optimized possibilistic detectors application module

After evolving a set of optimized PK-NN detectors using the GA, the software engineer could utilize them to detect smells and/or identify their types on unseen software systems as depicted by Figure 6. Indeed, the expert could utilize the best detector(s) or all of them. Each detector determines the similarity between the unseen software class and the software classes existing in the BE. Then, a number of the most similar software classes to the unseen class are chosen (for more details, please refer to Sub-section III-D). As the obtained detectors could generate different class labels, the majority voting strategy is adopted to have an effective decision. We recommend the software engineer to apply the identification process on the smelly classes that have been detected over the detection process.

IV. Experimental validation

To assess the performance of ADIPOK-UMT, we address the following RQs (Research Questions):

- To what extent does ADIPOK-UMT well-perform in the detection of smells under software metrics uncertainty?
- How is the performance of ADIPOK-UMT in the identification of smell types under software metrics uncertainty?

A. Software Systems studied

The context of our work consists on six commonly used Java software projects, which are ArgoUML[3], Azureus[4], Xerces-J[5], GanttProject[6], Apache Ant[7], and JFreechart[8]. ArgoUML is a well-known UML modeling tool. Xerces-J is a popular system for parsing XML files. GanttProject is an open source cross-platform for project management. Apache Ant is a build automation system for the Java applications. Azureus is a client program devoted to sharing files. JFreechat is a powerful Java open-source library that displays professional graphical charts. Table IV reports some descriptive statistics about the considered systems. The choice of these software systems is justified mainly by the following two reasons: (1) These systems are publicly available as well their code sources, and (2) they are widely used by the SBSE community over their experimental studies as they are rich in terms of code smells. In this study, we used the holdout cross-validation strategy for the assessment of our approach. As shown by Figure 7, the dataset is divided into 70% of the BE for the training set and 30% of the dataset for the test set. The former is used in the training phase while the test set

[3]http://argouml.tigris.org/
[4]http://vuze.com/
[5]http://xerces.apache.org/xerces-j/
[6]https://sourceforge.net/projects/ganttproject/files/OldFiles/
[7]http://ant.apache.org
[8]http://www.jfree.org/jfreechart/

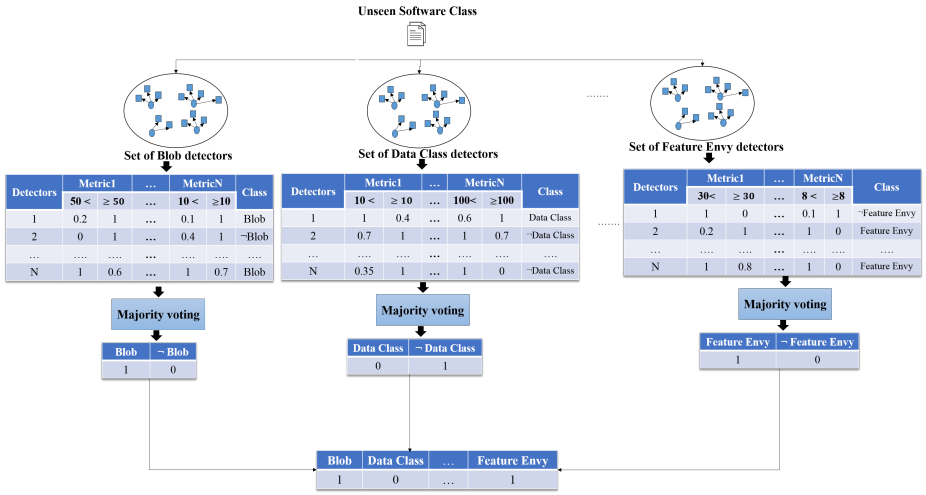

Fig. 6. Illustration of the use of ADIPOK-UMT on the identification case.

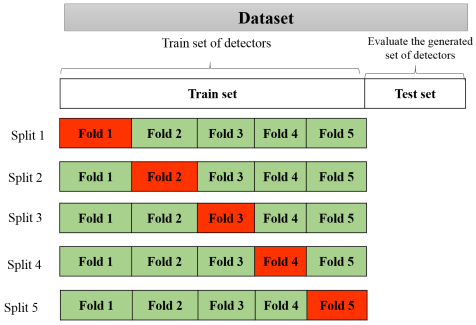

Fig. 7. Illustration of the training and the evaluation phases for the ADIPOK-UMT approach.

TABLE IV
USED SOFTWARE IN THE EXPERIMENTATION

Systems	Version	NOC	KLOC
GanttProject	1.10.2	245	41
ArgoUML	0.19.8	200	300
Xerces-J	2.7.0	991	240
JFreeChart	1.0.9	521	170
Apache Ant	1.7.0	1,839	327
Azureus	2.3.0.6	1,449	42

(i.e., instances not used in the training set) is adopted for the evaluation process. It is worth noting that in the training phase, we applied the five-folds cross-validation in the aim to train the classifiers (cf. Figure 7). Therefore, the training set is divided into five chunks. Each chunk is considered as test data and four times as training data. At the end, the obtained metrics results on the test data are averaged over the five chunks.

B. Baselines under comparison

To compare our proposed approach with some of the relevant state-of-the-art approaches, three relevant baselines were chosen: DECOR [21], GP [22], and BLOP [23]. The following is a brief summary of each baseline approach's working principle:

- DECOR: This approach employs a set of predefined detection rules (called Rule cards) for the detection of code smells.
- GP: This approach uses IF-THEN rules that are generated from a BE (that encompasses code smells) using the Genetic Programming.
- BLOP: This approach has tackled the problem of code smell detection as a Bilevel optimization where the upper level contains the detection rules that are evolved in the aim to detect artificial code smells that are generated by the lower level.

C. Evaluation metrics and statistical testing

To compare the performance obtained by the different methods, we employ two classification performance metrics that are insensitive to the difference size between smelly classes and non-smelly ones. The first one is the *F-measure*, previously detailed in Sub-section III-D. The evaluation of the methods should not rely solely on the *F-measure* as it could be biased too. Based on this fact, we also calculate as a second metric, the Matthews Correlation Coefficient (*MCC*) (see Equation 6) [24]. This metric has been shown to be insensitive to the bias of the imbalance size of classes and report an accurate description of the confusion matrix [17].

As all the considered methods for the comparison have stochastic nature, except DECOR and the greedy PK-NN, the generated results could change from one run to another one on the same experiment. To cope with such stochastic nature, we employ the Wilcoxon test [25] in a pairwise fashion with a significance level of 5%. Thus, every experiment is performed 31 times and then the medians' values are obtained and analysed to detect the performance differences among the compared methods. However, the Wilcoxon test is not eligible to produce any clearer and precise interpretation regarding the difference magnitude. Therefore, we adopted the Vargha-Delaney A measure [26] for the evaluation of the difference magnitude since it is a non-parametric effect size measure.

TABLE V
PARAMETER SETTINGS

Parameters	ADIPOK-UA	GP	BLOP	PK-NN
Crossover rate	0.9	0.9	0.8	0.8
Mutation rate	0.1	0.5	0.5	0.05
Population size	200	100	30	50

The A measure could be divided into three levels: (1) small if A is less than 0.64 or greater than 0.64, (2) medium if A is less than 0.36 or greater than 0.64, and (3) large if A is less than 0.29 or greater than 0.71.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \quad (6)$$

D. Parameter configurations

An important problem that should be taken into account before lunching search algorithms is the parameter settings. Indeed, the parameters' values have an important influence on the performance. To manage this problem, we utilize the trial-and-error method [27] to configure the parameters' values of ADIPOK-UMT and PK-NN. The other search methods' parameters are settled according to their original papers (cf. Table V).

For the fairness of comparison, the same stopping criterion is utilized for all search algorithms, which corresponds to 256, 500 fitness evaluations. We highlight that the execution of the different search algorithms is stopped as soon as the stopping criterion is met.

V. ANALYSIS OF THE RESULTS

A. Results for RQ1

Table VI presents the metrics' results of the five-peer detection methods on the six different software systems. Over this study, we need to prove that ADIPOK-UMT performs well for both environments: (1) with uncertain uncertain metrics' thresholds values and (2) without uncertainty. For this reason, we utilized two levels of uncertainty. The first level is expressed with the rate of uncertain attributes in the dataset $P \geqslant 50\%$; while for the second level, P=0%. According to this table, ADIPOK-UMT outperforms all the remaining methods for the identification process under uncertainty with an *F-measure* lying between 90.60 and 92.77. The second-best method is PK-NN with an *F-measure* varying between 34.19 and 50.46. The three other methods DECOR, GP, and BLOP, provide poor results with maximum *F-measure* values of 27.40, 37.22, and 48.30, respectively. The outperformance of ADIPOK-UMT and PK-NN over GP and BLOP could be explained by the fact that these two latter methods do not take into consideration the uncertainty factor when building their classifiers, which is not the case of ADIPOK-UMT and PK-NN. As we mentioned previously, the uncertainty lies in the attribute values of the BE. Indeed, the ADIPOK-UMT distance measure is the Affinity (see section II), which is well-suited distance measure for computation of the similarity between software classes having uncertain metrics' thresholds values. Moreover, the ADIPOK-UMT fitness function is the *F-measure*, which is insensitive to the problem of different sizes between two classes. However, PK-NN is evaluated based on the PCC (Percentage of correct classification), which is not a good choice for the case of imbalanced data class. On the other hand, ADIPOK-UMT carries out feature (metric) selection in addition to the optimization of the K parameter. In contrast, PK-NN method optimizes only the K parameter. Moreover, our proposed approach performs in an implicitly the weighting of software metrics (features) with respect to their computed discretization threshold. Such fact is not performed by the remaining approaches (including the baseline PK-NN). DECOR results are the poorest ones 08.60 and 27.40; due to the fact that the rules are manually conceived without considering the uncertainty aspect lying in the BE. Similar results are accomplished for the *MCC* metric since all methods except ours are sensitive to the problem of imbalanced data. Table VI presents the achieved A statistic results of ADIPPK-UA, DECOR, GP, BLOP, and PK-NN based on the F-measure and the MCC measures. One can see that ADIPOK-UMT achieves an A value higher than 0.89 (large) on the set of chosen software projects in the experimental study.

Table VII displays the metrics' results for the five-peer detection methods for the case of certain environment (P=0%). In such case, the BE is certain as the metrics' thresholds values are certain. Based on this table, ADIPOK-UMT method performs better than the remaining methods with an *F-measure* rate spans between 92.03 and 94.86. The second-best method is BLOP with an *F-measure* rate laying between 32.10 and 55.80. The three remaining methods DECOR, GP, and PK-NN, generate poor results with maximum *F-measure* rate values 35.86, 40.56, and 52.36, respectively. The outperformance of ADIPOK-UMT method over the four considered peer methods could be justified by the fact that the ADIPOK-UMT fitness function is a well-suited metric to the data imbalance problem. For the *MCC* metric, the presented results are similar to those of the *F-measure*. Indeed, the good results reported by DECOR, GP, BLOP, and PK-NN are biased and usually generated by haphazardly since they are not able to treat the data imbalance issue over the learning phase. The worst results belong to DECOR method due to the fact that its detection rules are manually predefined by a software engineer. Therefore, they are not effective for the case of imbalance class sizes. Contrariwise, ADIPOK-UMT has proven its outperformance mainly thanks to the insensitivity of its fitness function to the problem of imbalanced data. Table VII shows the achieved A statistic results of the five approaches under comparison. One can see that ADIPOK-UMT succeeds to obtain an A value greater than 0.82 (large) based on the F-measure and the MCC measures on the set of chosen software projects in the experimental study.

In nutshell, ADIPOK-UMT has shown great performance comparing to the remaining peer methods for both environments: (1) uncertain environment (with uncertain metrics' thresholds values), and (2) certain one (charecterized only by

certain metrics' thresholds values).

B. Results for RQ2

Table VIII reports the results of the considered metrics for the five peer identification methods on the eight smell types. The identification process is an arduous task comparing to the detection one since the imbalance data ratio is greater than the detection one. In this part, we need to evaluate the performance of our ADIPOK-UMT against the remaining approaches for the identification of code smell types under uncertain (P$\geqslant$50%) and certain (P=0%) environments. From Table VIII, we can observe that AIPOK-UA outperforms the remaining methods with an *F-measure* laying between 83 and 89.15. The second best method is PK-NN with an *F-measure* varying between 30.71 and 42.03. The three remaining methods, DECOR, GP and BLOP, show bad results with maximum *F-measure* values of 21.30, 30.61, and 41.18, respectively. The significant outperformance of ADIPOK-UMT over DECOR, GP and BLOP methods could be explained by the fact that: (1) these methods do not consider the uncertain metrics' thresholds values during the learning process and (2) they are not well-suited to manage the data imbalance problem. They only discard and/or replace the uncertain parts which causes a loss of information and consequently generate ineffective detectors (classifiers). In contrast, the performance of ADIPOK-UMT methods could be explained by the fact that it manages uncertain metrics' thresholds values by using an adequate distance measure (affinity (cf. Equation 4)) to measure the similarity between the software classes. Moreover, ADIPOK-UMT is able to deal with the problem of data imbalance as it uses the *F-measure*, which is known by its insensitivity to such problem. Additionally, our proposed approach performs the weighting of chosen metrics' thresholds in an implicit manner using the possibility degrees, which aid to well predict the class labels of the software systems. The ADIPOK-UMT superiority could be justified by three reasons. The first reason is that ADIPOK-UMT method carries out feature (quality metrics) selection and optimizes the K parameter while the PK-NN optimizes only the K value. The second reason is that the chosen metrics are weighted after discretization through the possibility distribution-based encoding. The third reason is that the PK-NN does not take into account the uncertainty and the data imbalance issue over the learning process. This could be justified by the employment of unsuitable evaluation metric. The DECOR has shown poorest results comparing to the other methods with an *F-measure* varying between 04.65 and 21.30; such poorer results are obtained since the detection rules are manually fixed by a software engineer without giving any importance to the problems mentioned previously. The same results are obtained for the *MCC* metric as it is insensitive to the data imbalance problem. The similarity distance measure (affinity measure) and the used metrics (*F-measure* and *MCC*) make the ADIPOK-UMT well-suited to handle both environments considered in this work. Table VIII presents the achieved A statistic results of ADIPE, DECOR, GP, MOGP, and BLOP using the F-measure and the MCC metrics for code smell types identification case in an uncertain environment. The achieved results reveal that ADIPK-UA succeeds in reaching a value greater than 0.88 (large) over the all considered smell types. Therefore, we can deduce that our approach significantly surpasses its competitors.

Table IX presents the metrics' results of the five peer methods over the smell type identification process for the case when there is only certain metrics' values (P = 0%). We recall that in the certain environment, the metrics' thresholds values of the BE are certain. As there is no uncertainty, the only important problem that remains the high imbalance data ratio. According to Table IX, our ADIPOK-UMT performs well than the other methods with an *F-measure* value spanning between 83.98 and 90.62. The second best-method is BLOP with an *F-measure* laying between 20.74 and 43.22. The two remaining methods, GP, and PK-NN, provide bad results with maximum *F-measure* values of 33.16 and 40.26, respectively. The outperformance of ADIPOK-UMT over these methods could be justified by the fact that the ADIPOK-UMT approach is able to deal with the problem of imbalanced data thanks to the adopted fitness function. We recall that the data imbalance in the identification problem is higher than the detection problem. The poorest results remain to the DECOR method, as its detection rules are manually conceived which is not effective for the case of data imbalance problem. For the *MCC* metric, the reported results are similar to those of *F-measure*. Table IX illustrates the achieved A statistic results of the five algorithms under comparison based on the F-measure and the MCC metrics for the identification of the eight smell types under a certain environment. The illustrated results show that ADIPOK-UMT succeeds in reaching a value greater than 0.88 (large) over all the chosen smell types. Thus, we can deduce that ADIPOK-UMT significantly surpasses DECOR, GP, BLOP, and PK-NN. Indeed, the good results shown by some considered methods are incorrect and in most times are generated haphazardly, as these methods are not adequate for dealing with uncertain and data imbalance issues. Contrariwise, ADIPOK-UMT has shown great performance when handling the imbalance data and the uncertain attributes' values issues thanks to the strength and the insensitivity of its fitness function *F-measure* as well as the used similarity distance used to determine the similarity between the labelled software classes and the unseen ones.

VI. Related works

Existing code smell detection approaches could be classified based on their employed techniques into four categories: Rule/heuristic-based category, Search-based category, Machine Learning-based category, and Others. Nonetheless, there is no attempt to identify code smells under uncertain environment. The first attempt in the detection of code smells was conducted in the Rule/heuristic-based category, where the methods utilize known (or/and newly) metrics to detect the violated design practices. Erni and Lewrentez [28] proposed a composition of m-tuple of different metrics (multi-metrics) to assess the frameworks' performance in the aim to enhance them. The

TABLE VI

F-measure AND *MCC* MEDIAN SCORES OF ADIPOK-UMT, DECOR, GP, BLOP, AND PK-NN FOR 31 RUNS OF THE DETECTION PROCESS AT AN UNCERTAIN ATTRIBUTE LEVEL P$\geqslant$50%.

Systems	ADIPOK-UA		DECOR		GP		BLOP		PK-NN	
	F-measure	MCC	F-measure	MCC	F-measure	MCC	F-measure	MCC	F-measure	MCC
Gantt Project	92.77 (+ + + +) (l l l l)	94 (+ + + +) (l l l l)	27.40 (+ + +) (m l l)	24.56 (+ + +) (m l l)	37.22 (+ +) (m m)	35.63 (+ +) (m m)	48.30 (-) (s)	46.91 (-) (s)	50.46	49.12
Argo UML	92.05 (+ + + +) (l l l l)	93.88 (+ + + +) (l l l l)	25.60 (+ + +) (m l l)	22 (+ + +) (m l l)	35.09 (+ +) (m m)	33.26 (+ +) (m m)	46 (-) (s)	44.23 (-) (s)	50.06	47.78
Xerc ess-J	90.81 (+ + + +) (l l l l)	91.45 (+ + + +) (l l l l)	20.81 (+ + +) (m l l)	17.42 (+ + +) (m l l)	29.85 (+ +) (m m)	26.42 (+ +) (m m)	42.87 (-) (s)	38.56 (+) (m)	46.43	44.12
JFree Chart	93.24 (+ + + +) (l l l l)	95.33 (+ + + +) (l l l l)	14.30 (+ + +) (m l l)	11.70 (+ + +) (m l l)	25.36 (+ +) (m l)	21.79 (+ +) (m l)	35.18 (+) (m)	31 (+) (m)	45.50	41.82
Azu reus	90.87 (+ + + +) (l l l l)	92.41 (+ + + +) (l l l l)	10.09 (+ + +) (m l l)	08.39 (+ + +) (m l l)	18.50 (+ +) (m l)	15.26 (+ +) (m l)	28.72 (+) (m)	26.40 (+) (m)	39.66	36.12
Apache Ant	90.60 (+ + + +) (l l l l)	92.18 (+ + + +) (l l l l)	08.60 (+ + +) (m l l)	05.59 (+ + +) (m l l)	14.37 (+ +) (m l)	12 (+ +) (m l)	22.34 (+) (m)	21.07 (+) (m)	34.19	30.28

The sign "+" at the i^{th} position signifies that the algorithm *F-measure*e (or *MCC*) median value is statically distinct from the i^{th} algorithm value. The sign "-" signifies the opposite. Likewise, the effect sizes values (small (s), medium (m), and large (l)) based on the A-statistics are given. Best *F-measure* (or *MCC*) values are in Bold. Second-best *F-measure* (or *MCC*) values are underlined.

TABLE VII

F-measure AND *MCC* MEDIAN SCORES OF ADIPOK-UMT, DECOR, GP, BLOP, AND PK-NN FOR 31 RUNS OF THE DETECTION PROCESS AT AN UNCERTAIN ATTRIBUTE LEVEL P=0%.

Systems	ADIPOK-UA		DECOR		GP		PK-NN		BLOP	
	F-measure	MCC	F-measure	MCC	F-measure	MCC	F-measure	MCC	F-measure	MCC
Gantt Project	93.98 (+ + + +) (l l l l)	95.62 (+ + + +) (l l l l)	35.34 (+ + +) (m l l)	30.10 (+ + +) (m l l)	40.56 (+ +) (m m)	37.02 (+ +) (m m)	52.36 (-) (s)	48.13 (-) (s)	55.80	52.74
Argo UML	93.22 (+ + + +) (l l l l)	94.91 (+ + + +) (l l l l)	35.86 (- + +) (s m m)	29.06 (+ + +) (m m l)	36.04 (+ +) (m l)	34.50 (+ +) (m l)	51.07 (+) (m)	47.21 (+) (m)	57.78	55.16
Xerc ess-J	92.03 (+ + + +) (l l l l)	93.72 (+ + + +) (l l l l)	19.71 (+ + +) (m l l)	12.58 (+ + +) (m l l)	32.04 (+ +) (m l)	29.20 (+ +) (m l)	45 (+) (m)	41.83 (+) (m)	51.42	49.98
JFree Chart	94.86 (+ + + +) (l l l l)	95.52 (+ + + +) (l l l l)	17.08 (+ + +) (m l l)	11.40 (+ + +) (m l l)	30.63 (+ +) (m m)	26.58 (+ +) (m l)	38.33 (+) (m)	33.16 (+) (m)	50.03	47.66
Azu reus	93.12 (+ + + +) (l l l l)	94.42 (+ + + +) (l l l l)	12.97 (+ + +) (m l l)	09.88 (+ + +) (m l l)	24.15 (+ +) (m m)	20.01 (+ +) (m l)	32.89 (+) (m)	28.39 (+) (m)	42.33	40.82
Apache Ant	93.95 (+ + + +) (l l l l)	95.30 (+ + + +) (l l l l)	11.20 (+ + +) (m l l)	06.13 (+ + +) (m l l)	22.53 (+ +) (m m)	17.60 (+ +) (m m)	31.59 (-) (s)	27.15 (-) (s)	32.10	31.06

The sign "+" at the i^{th} position signifies that the algorithm *F-measure*e (or *MCC*) median value is statically distinct from the i^{th} algorithm value. The sign "-" signifies the opposite. Likewise, the effect sizes values (small (s), medium (m), and large (l)) based on the A-statistics are given. Best *F-measure* (or *MCC*) values are in Bold. Second-best *F-measure* (or *MCC*) values are underlined.

detection strategy proposed by Lanza and Marinescu [29] formulates rules in the form of metric-threshold combinations that capture the detours from good design practices and heuristics. Moha et al., [21] have proposed a tool called DECOR that detects code smells based on “Rule cards”, which include the defect symptoms and its properties.

The machine-learning based category has obtained much attention over the last years. The methods belonging to this

TABLE VIII
F-measure AND *MCC* MEDIAN SCORES OF ADIPOK-UMT, DECOR, GP, BLOP, AND PK-NN FOR 31 RUNS OF THE IDENTIFICATION PROCESS WITH AN UNCERTAIN ATTRIBUTE LEVEL P$\geqslant$50%.

Code smells	ADIPOK-UA		DECOR		GP		BLOP		PK-NN	
	F-measure	MCC	F-measure	MCC	F-measure	MCC	F-measure	MCC	F-measure	MCC
Blob	89.15 (+ + + +) (l l l l)	90.51 (+ + + +) (l l l l)	21.30 (+ + +) (m l l)	19.08 (+ + +) (m l l)	30.61 (+ +) (m m)	28 (+ +) (m m)	41.18 (-) (s)	38.70 (-) (s)	42.03	42
Data Class	88.60 (+ + +) (l l l)	90.85 (+ + +) (l l l)	N/A	N/A	24.13 (+ +) (m m)	23.48 (+ +) (m m)	35.60 (-) (s)	33.45 (+) (m)	40.44	40.12
Feature Envy	86.97 (+ + +) (l l l)	88.77 (+ + +) (l l l)	N/A	N/A	24.12 (+ +) (m m)	23.73 (+ +) (m m)	35.76 (-) (s)	34.12 (-) (s)	39.56	39.98
Long Method	85.16 (+ + +) (l l l)	88.86 (+ + +) (l l l)	N/A	N/A	22.35 (+ +) (m l)	20 (+ +) (m l)	33.50 (-) (s)	31.20 (+) (m)	38.11	37.25
Duplicate Code	84.92 (+ + +) (l l l)	86.82 (+ + +) (l l l)	N/A	N/A	18.25 (+ +) (m l)	16.90 (+ +) (m l)	31 (+) (m)	30.68 (+) (m)	36.08	35.74
Long Parameter List	86.97 (+ + +) (l l l)	87.74 (+ + +) (l l l)	N/A	N/A	13.31 (+ +) (m l)	11.16 (+ +) (m l)	28.11 (+) (m)	27.84 (+) (m)	35.82	33
Spaghetti Code	83.45 (+ + + +) (l l l l)	85.15 (+ + + +) (l l l l)	06.64 (- + +) (s l l)	05.14 (- + +) (s l l)	10.71 (+ +) (m l)	09.32 (+ +) (m l)	24.18 (+) (m)	23.65 (+) (m)	32.3	31.48
Functional Decomposition	83 (+ + + +) (l l l l)	85.22 (+ + + +) (l l l l)	04.65 (- + +) (s m l)	04 (- + +) (s m l)	08.50 (+ +) (m l)	08.21 (+ +) (m l)	19.72 (+) (m)	18.05 (+) (m)	30.71	29.77

The sign "+" at the i^{th} position signifies that the algorithm *F-measuree* (or *MCC*) median value is statically distinct from the i^{th} algorithm value. The sign "-" signifies the opposite. Likewise, the effect sizes values (small (s), medium (m), and large (l)) based on the A-statistics are given. Best *F-measure* (or *MCC*) values are in Bold. Second-best *F-measure* (or *MCC*) values are underlined.

category utilize machine learning techniques to detect code smells. These methods are learned using training data and then tested to detect the deviation from the good design classes. Different learning approaches have been used in the detection of code smells, including decision tree (DT) [30]. The author used DT to predict anti-patterns in software projects. The Bayesian Networks have been implemented by Khomh et al. [31], [32] to detect the occurrences of code smells on software projects. The Support Vector Machine (SVM) was used by Maiga et al., [33], [34] to propose an approach called SVMDetect to detect code smells. To investigate the performance of different machine learning techniques, Fontana et al. [35] employed 16 supervised machine learning algorithms with their boosting variants for the code smell detection. The code smell detection problem was seen by Guggulothu and Abdul Moiz [36] as a Multi-label classification problem as a software class could include more than a single smell type. Recently, some works [37], [38] used the Deep learning techniques that take as input different information types (e.g. structural information, semantic information, historical information) to detect code smells.

In the search-based category, the existing methods have adopted different meta-heuristic algorithms (e.g. Genetic Programming (GP), Genetic Algorithm (GA), and so one) to resolve the optimization issues. For instance, the GP was adopted by Ouni et al., [22] in the aim to automatically derive detection rules from badly designed examples. Sahin et al., [23] was the first to integrate the bi-level optimization problem in their Bi-Level Optimization Problem (BLOP) approach to generate a set of detection rules. This approach is composed of two levels; the upper-level is charged by the generation of detection rules in order to cover not only the maximum code smell examples but also the artificial ones obtained from the second level; while the lower-level is responsible for exploring the non-detected smells by the created rules from the upper level. In [39], the authors generate the detection rules based on the multi-objective aspect. Their proposed approach called Multi-Objective Genetic Programming (MOGP) that explores the search space to find the best rules to enhance the set of detected defect examples and decrease the number of detected well-designed examples.

The last category includes the methods that are based on historical information concerning the change of code smells over different software versions [40]–[42] and also the methods that are based on visualization to detect code smells over code fragments [43]–[45].

TABLE IX
F-measure AND *MCC* MEDIAN SCORES OF ADIPOK-UMT, DECOR, GP, BLOP, AND PK-NN FOR 31 RUNS OF THE IDENTIFICATION PROCESS WITH AN UNCERTAIN ATTRIBUTE LEVEL P=0%.

Code smells	ADIPOK-UA		DECOR		GP		PK-NN		BLOP	
	F-measure	MCC	F-measure	MCC	F-measure	MCC	F-measure	MCC	F-measure	MCC
Blob	90.62 (+ + + +) (l l l l)	92.25 (+ + + +) (l l l l)	30.6 (- + +) (s m m)	28.04 (- + +) (s m m)	33.16 (+ +) (m m)	30.25 (+ +) (m m)	40.26 (-) (s)	37.60 (-) (s)	43.22	41.72
Data Class	88.93 (+ + +) (l l l)	89.23 (+ + +) (l l l)	N/A	N/A	26.7 (- +) (s m)	24.19 (- +) (s m)	30.88 (+) (m)	29.08 (+) (m)	37.13	35.10
Feature Envy	87 (+ + +) (l l l)	88.92 (+ + +) (l l l)	N/A	N/A	25.16 (- +) (s m)	24.71 (- +) (s m)	29.60 (+) (m)	28.33 (+) (m)	36.75	35.04
Long Method	86.77 (+ + +) (l l l)	89.16 (+ + +) (l l l)	N/A	N/A	23.92 (- +) (s m)	22.59 (- +) (s m)	28.11 (+) (m)	26.38 (+) (m)	34	31.92
Duplicate Code	85.14 (+ + +) (l l l)	87.02 (+ + +) (l l l)	N/A	N/A	19.64 (+ +) (m m)	17 (+ +) (m m)	26.7 (+) (m)	24.61 (+) (m)	32.70	31.11
Long Parameter List	86.53 (+ + +) (l l l)	88.42 (+ + +) (l l l)	N/A	N/A	14.12 (+ +) (m m)	13.74 (+ +) (m m)	22.04 (+) (m)	21.8 (+) (m)	29.92	29
Spaghetti Code	84.12 (+ + + +) (l l l l)	86.35 (+ + + +) (l l l l)	08.13 (- + +) (s m m)	06.90 (- + +) (s m m)	11.30 (+ +) (m m)	11 (+ +) (m m)	20.6 (+) (m)	20.02 (+) (m)	25.36	24.78
Functional Decomposition	83.98 (+ + + +) (l l l l)	85.60 (+ + + +) (l l l l)	06.35 (- + +) (s m m)	04.15 (- + +) (s m m)	10.22 (+ +) (m m)	10.09 (+ +) (m m)	14.36 (+) (m)	13.87 (+) (m)	20.74	20.16

The sign "+" at the i^{th} position signifies that the algorithm *F-measure*e (or *MCC*) median value is statically distinct from the i^{th} algorithm value. The sign "-" signifies the opposite. Likewise, the effect sizes values (small (s), medium (m), and large (l)) based on the A-statistics are given. Best *F-measure* (or *MCC*) values are in Bold. Second-best *F-measure* (or *MCC*) values are underlined.

VII. CONCLUSION AND FUTURE WORKS

In this work, we have proposed ADIPOK-UMT as a new method to detect and/or identify code smells. In this paper, we have considered an important problem that is usually neglected (or ignored) by the SBSE community, which is the uncertain metrics' thresholds values of the detection rules. The appearance of such kind of uncertainty could be explained by the fact that the experts are not able to give precise metric values in each rule to detect code smells. The ignoring of the uncertainty causes a loss of information and consequently inaccurate results. Our approach ADIPOK-UMT, consisting of a GA that evolves a population of PK-NN based on a PBE characterized by uncertain metrics' thresholds values, has proven superiority over the four-peer methods thanks to three important merits. First, the utilization of feature selection task in addition to the optimization of K parameter. Secondly, the adoption of the possibility distribution-based encoding that permits the weighting of the software metrics. Thirdly, the use of adequate similarity measure (affinity) to measure the similarity between instances according to their uncertain metrics' thresholds values. Fourthly, the adoption of the *F-measure* as a fitness function, which is known by its ability for being insensitive to another important problem called data imbalance problem. It is worth noting that the ratio of the data imbalance problem is higher for the identification problem comparing to the detection one. As a result, the statistical analysis of the generated results has proven the competitiveness and the strength of our proposal comparing to the four relevant state-of-the-art methods.

Following this work, we have detected a number of interesting perspectives. First, detecting and identifying code smells are fastidious task for human experts since there are no exact rules. Therefore, the uncertainty could be located at the class labels [46] as well as the specification of the metrics' thresholds values. Hence, it would be interesting to handle the uncertainty of class labels and metrics' thresholds values. Second, in a real world situation, the number of unlabelled instances exceeds the number labelled ones. Thus, another path for future investigation would be the use of a semi-supervised technique to to mimic the real world situations. Finally, it would be challenging to apply ADIPOK-UMT to detect and identify code smells in an uncertain environment on web services and mobile-based applications [47], [48].

REFERENCES

[1] E. V. de Paulo Sobrinho, A. De Lucia, and M. de Almeida Maia, "A systematic literature review on bad smells—5 w's: which, when, what, who, where," *IEEE Transactions on Software Engineering*, 2018.

[2] M. I. Azeem, F. Palomba, L. Shi, and Q. Wang, "Machine learning techniques for code smell detection: A systematic literature review and meta-analysis," *Information and Software Technology*, vol. 108, pp. 115–138, 2019.

[3] F. Sabir, F. Palma, G. Rasool, Y.-G. Guéhéneuc, and N. Moha, "A systematic literature review on the detection of smells and their evolution in object-oriented and service-oriented systems," *Software: Practice and Experience*, vol. 49, no. 1, pp. 3–39, 2019.

[4] M. Fowler and K. Beck, *Refactoring: Improving the Design of Existing Code*. Addison-Wesely, 1999.

[5] A. S. Cairo, G. d. F. Carneiro, and M. P. Monteiro, "The impact of code smells on software bugs: A systematic literature review," *Information*, vol. 9, no. 11, p. 273, 2018.

[6] H. Liu, Q. Liu, Z. Niu, and Y. Liu, "Dynamic and automatic feedback-based threshold adaptation for code smell detection," *IEEE Transactions on Software Engineering*, vol. 42, no. 6, pp. 544–558, 2015.

[7] M. S. Boutaib and Z. Elouedi, "Incremental possibilistic decision trees in non-specificity approach," in *Data Science and Knowledge Engineering for Sensing Decision Support: Proceedings of the 13th International FLINS Conference (FLINS 2018)*, vol. 11. World Scientific, 2018, p. 339.

[8] L. A. Zadeh, "Fuzzy sets," *Information and control*, vol. 8, no. 3, pp. 338–353, 1965.

[9] Z. Pawlak, "Rough sets," *International journal of computer & information sciences*, vol. 11, no. 5, pp. 341–356, 1982.

[10] P. Smets and R. Kennes, "The transferable belief model," *Artificial intelligence*, vol. 66, no. 2, pp. 191–234, 1994.

[11] R. Hentech, I. Jenhani, and Z. Elouedi, "Possibilistic airs induction from uncertain data," *Soft Computing*, vol. 20, no. 1, pp. 3–17, 2016.

[12] I. Jenhani, N. B. Amor, Z. Elouedi, S. Benferhat, and K. Mellouli, "Information affinity: A new similarity measure for possibilistic uncertain information," in *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, vol. 4724. Springer, 2007, pp. 840–852.

[13] R. Krishnapuram and J. M. Keller, "The possibilistic c-means algorithm: insights and recommendations," *IEEE transactions on Fuzzy Systems*, vol. 4, no. 3, pp. 385–393, 1996.

[14] M. Krętowski and M. Grześ, "Global learning of decision trees by an evolutionary algorithm," in *Information Processing and Security Systems*. Springer, 2005, pp. 401–410.

[15] A. Samet, E. Lefèvre, and S. B. Yahia, "Evidential data mining: precise support and confidence," *Journal of Intelligent Information Systems*, vol. 47, no. 1, pp. 135–163, 2016.

[16] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[17] A. Barbez, F. Khomh, and Y.-G. Guéhéneuc, "A machine-learning based ensemble method for anti-patterns detection," *Journal of Systems and Software*, vol. 161, p. 110486, 2020.

[18] C. J. V. Rijsbergen, *Information Retrieval*, 2nd ed. USA: Butterworth-Heinemann, 1979.

[19] B. B, "Genetic algorithms for function optimization," Dissertation, University of Alberta, 1980.

[20] M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *computer*, vol. 27, no. 6, pp. 17–26, 1994.

[21] 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, 2009.

[22] 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.

[23] 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.

[24] B. W. Matthews, "Comparison of the predicted and observed secondary structure of t4 phage lysozyme," *Biochimica et Biophysica Acta (BBA)-Protein Structure*, vol. 405, no. 2, pp. 442–451, 1975.

[25] W. J. Conover and W. J. Conover, "Practical nonparametric statistics," 1980.

[26] A. Vargha and H. D. Delaney, "A critique and improvement of the cl common language effect size statistics of mcgraw and wong," *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000.

[27] A. E. Eiben and S. K. Smit, "Parameter tuning for configuring and analyzing evolutionary algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 19–31, 2011.

[28] K. Erni and C. Lewerentz, "Applying design-metrics to object-oriented frameworks," in *Proceedings of the 3rd international software metrics symposium*. IEEE, 1996, pp. 64–74.

[29] M. Lanza and R. Marinescu, *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. Springer Science & Business Media, 2007.

[30] J. Kreimer, "Adaptive detection of design flaws," *Electronic Notes in Theoretical Computer Science*, vol. 141, no. 4, pp. 117–136, 2005.

[31] 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.

[32] ——, "BDTEX: A GQM-based Bayesian approach for the detection of antipatterns," *Journal of Systems and Software*, vol. 84, no. 4, pp. 559–572, 2011.

[33] 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.

[34] A. Maiga, N. Ali, N. Bhattacharya, A. Sabané, Y.-G. Guéhéneuc, G. Antoniol, and E. Aïmeur, "Support vector machines for anti-pattern detection," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering,*. IEEE, 2012, pp. 278–281.

[35] F. A. Fontana, M. V. Mäntylä, M. Zanoni, and A. Marino, "Comparing and experimenting machine learning techniques for code smell detection," *Empirical Software Engineering*, vol. 21, no. 3, pp. 1143–1191, 2016.

[36] T. Guggulothu, "Code smell detection using multilabel classification approach," *arXiv preprint arXiv:1902.03222*, 2019.

[37] M. Hadj-Kacem and N. Bouassida, "Deep representation learning for code smells detection using variational auto-encoder," in *International Joint Conference on Neural Networks,*. IEEE, 2019, pp. 1–8.

[38] H. Liu, J. Jin, Z. Xu, Y. Bu, Y. Zou, and L. Zhang, "Deep learning based code smell detection," *IEEE Transactions on Software Engineering*, pp. 1–28, 2019.

[39] 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.

[40] D. Rapu, S. Ducasse, T. Gîrba, and R. Marinescu, "Using history information to improve design flaws detection," in *Proceedings of the 8th European Conference on Software Maintenance and Reengineering,*. IEEE, 2004, pp. 223–232.

[41] F. Palomba, G. Bavota, M. D. Penta, R. Oliveto, A. D. Lucia, and D. Poshyvanyk, "Detecting bad smells in source code using change history information," in *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering,*, IEEE. IEEE, 2013, pp. 268–278.

[42] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, D. Poshyvanyk, and A. De Lucia, "Mining version histories for detecting code smells," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 462–489, 2014.

[43] E. V. Emden and L. Moonen, "Java quality assurance by detecting code smells," in *Proceedings of the 9th Working Conference on Reverse Engineering,*, IEEE. IEEE, 2002, pp. 97–106.

[44] G. Langelier, H. Sahraoui, and P. Poulin, "Visualization-based analysis of quality for large-scale software systems," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering,*. ACM, 2005, pp. 214–223.

[45] K. Dhambri, H. Sahraoui, and P. Poulin, "Visual detection of design anomalies," in *Proceedings of the 12th European Conference on Software Maintenance and Reengineering,*. IEEE, 2008, pp. 279–283.

[46] S. Boutaib, M. Elarbi, S. Bechikh, C.-C. Hung, and L. Ben Said, "Software anti-patterns detection under uncertainty using a possibilistic evolutionary approach," in *Proceedings of the 24th European Conference on Genetic Programming*. Springer Nature, 2021, pp. 181–197.

[47] N. Bessghaier, A. Ouni, and M. W. Mkaouer, "On the diffusion and impact of code smells in web applications," in *International Conference on Services Computing*. Springer, 2020, pp. 67–84.

[48] I. Saidani, A. Ouni, and M. W. Mkaouer, "Web service api anti-patterns detection as a multi-label learning problem," in *International Conference on Web Services*. Springer, 2020, pp. 114–132.