MEG: Multi-objective Ensemble Generation for Software Defect Prediction

Rebecca Moussa, Giovani Guizzo, Federica Sarro
Department of Computer Science, University College London (UCL)
United Kingdom
{rebecca.moussa.18,g.guizzo,f.sarro}@ucl.ac.uk

ABSTRACT

Background: Defect Prediction research aims at assisting software engineers in the early identification of software defect during the development process. A variety of automated approaches, ranging from traditional classification models to more sophisticated learning approaches, have been explored to this end. Among these, recent studies have proposed the use of ensemble prediction models (i.e., aggregation of multiple base classifiers) to build more robust defect prediction models. **Aims:** In this paper, we introduce a novel approach based on multi-objective evolutionary search to automatically generate defect prediction ensembles. Our proposal is not only novel with respect to the more general area of evolutionary generation of ensembles, but it also advances the state-of-the-art in the use of ensemble in defect prediction. **Method:** We assess the effectiveness of our approach, dubbed as **Multi-objective Ensemble Generation** (MEG), by empirically benchmarking it with respect to the most related proposals we found in the literature on defect prediction ensembles and on multi-objective evolutionary ensembles (which, to the best of our knowledge, had never been previously applied to tackle defect prediction). **Result:** Our results show that MEG is able to generate ensembles which produce similar or more accurate predictions than those achieved by all the other approaches considered in 73% of the cases (with favourable large effect sizes in 80% of them). **Conclusions:** MEG is not only able to generate ensembles that yield more accurate defect predictions with respect to the benchmarks considered, but it also does it automatically, thus relieving the engineers from the burden of manual design and experimentation.

CCS CONCEPTS

• **Software and its engineering → Search-based software engineering**; **Software defect analysis**; • **Computing methodologies → Ensemble methods**.

KEYWORDS

Defect Prediction, Search-Based Software Engineering, Multi-Objective Optimisation, Hyper-Heuristic, Empirical Study

ESEM'22, September 19–23, 2022, Helsinki, Finland
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9427-7/22/09…$15.00
https://doi.org/10.1145/3544902.3546255

ACM Reference Format:
Rebecca Moussa, Giovani Guizzo, Federica Sarro. 2022. MEG: Multi-objective Ensemble Generation for Software Defect Prediction. In *Proceedings of the 16th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement.*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3544902.3546255

1 INTRODUCTION

As the world becomes more and more dependent on software, tasks such as detecting defects early in the development process become more essential and critical. Fixing software defects costs billions per year. The later a bug is found in the process, the more it costs to both its users, and the company providing the software [35].

Defect Prediction (DP) aims at assisting software engineers in the early identification of software defects during the development process, and ideally, before it is shipped to its customers. A variety of automated approaches, ranging from traditional classification models to more sophisticated learning approaches, have been explored to this end. Among these, recent studies have found the use of ensemble prediction models (i.e., aggregation of multiple base classifiers) to achieve more accurate results than those that would have been obtained by relying on a single classifier.

However, designing an ensemble requires a non-trivial amount of effort and expertise with respect to the choice of the set of base classifiers, their hyper-parameter tuning, and the choice of the strategy used to aggregate the predictions. An inappropriate choice of any of these aspects can lead to over- or under-fitting, thereby heavily worsening the performance of the ensemble.

Examining all possible combinations is not computationally affordable, as the search space is too large, and there is a strong interaction among these aspects, which cannot be optimized separately. Such a large search space makes Search-Based Software Engineering a suitable solution for the problem of automatically generating effective ensembles for DP.

In this paper, we propose a novel use of multi-objective evolutionary algorithms to automatically generate defect prediction ensembles. We dub our proposed approach **Multi-objective Ensemble Generation** (MEG).

MEG is novel with respect to the existing proposals in the more general area of evolutionary generation of ensembles, which are all based on *Pareto-ensemble generation* as opposed to the concept of *Whole-ensemble generation* we introduce herein. Moreover, our study is the first to investigate the effectiveness of evolutionary ensemble for defect prediction.

In order to assess the effectiveness of MEG, we conduct a large-scale empirical study by benchmarking it against traditional base classifiers (as a sanity check), against the state-of-the-art multi-objective ensemble approach proposed by Petrić et al. [53] which, to the best of our knowledge, is the only one to use a diversity measure in the ensemble Defect Prediction literature, and against DIVACE proposed by Chandra et al., which is considered a seminal work for ensemble generation in the Evolutionary Computation literature. These [53] [17] are the work most closely related to ours, which motivated us to compare MEG to them.

To assess the effectiveness of MEG we carried out a thorough large-scale empirical study involving a total of 24 real-world software versions and 16 cross-version defect prediction scenarios, assessed according to the latest best practice for the evaluation of defect prediction and search-based approaches [7, 51, 55, 64].

Our results show that MEG is able to generate ensembles with similar or more accurate predictions than those achieved by all the other approaches considered in 73% of the cases (with large effect sizes in 80% of them). Not only does MEG yield good results, but it also relieves the engineer from the error-prone, burdensome, and time-consuming task of manually designing and experimenting with different ensemble configurations in order to find an optimal one for the problem at hand.

The main contributions of our work are: (1) The proposal of MEG, a novel multi-objective approach for the automated generation of ensembles based on the concept of *whole-ensemble generation.* (2) A large-scale empirical evaluation of the effectiveness of MEG for defect prediction which involves eight open-source Java software systems for a total of 24 software versions and 16 cross-version defect prediction scenarios. (3) The comparison of MEG to baseline defect prediction classifiers, the more recent approach to build ensembles for defect prediction [53], and a state-of-the-art multi-objective Pareto-ensemble generation (which has never applied to defect prediction before) [17].

We make the source code of MEG publicly available to facilitate its uptake for both researchers and practitioners [49]. We also share a replication package to allow for reproduction and extension [49].

2 BACKGROUND

Given the multi-disciplinary nature of this work, this section provides some background on Defect Prediction, Ensemble Learning, and Multi-Objective Evolutionary Optimisation.

2.1 Defect Prediction

Models applied for defect prediction generally exploit past data about software modules in order to classify the latter as either defective or not. To this end, predictive models infer one aspect of the data (dependent variable) from some combination of other aspects of the data (independent variables). In the context of binary defect prediction, the dependent variable is denoted by whether the software module contains defects or not, whereas the independent variables may vary depending on the project and can be extracted from various aspects of the software (e.g., source code metrics, process metrics) [10, 21, 47, 54, 75]. The performance of a predictive model relies on both the modelling technique and the independent variables used. A great deal of predictive models have been investigated in the literature. This includes widely used Machine Learning techniques such as Decision Trees, Logistic Regression and Naive Bayes [16, 27, 36]. More advanced techniques such as ensemble learning and search-based approaches have also been successfully applied for predicting software defects [3, 15, 20, 29, 43, 48, 57, 61]. However, no previous study has investigated the application of search-based approaches for the generation of ensemble for defect prediction.

2.2 Ensemble Learning

Ensemble Learning is a technique used for building more robust machine learning models achieving better performance by the means of combining multiple classifiers trained to solve the same problem.

Base models can be used to design more complex models by combining a number of them according to a given ensemble learning approach. The aim is to achieve a more robust model able to reduce both (1) the bias error, which arises from erroneous assumptions made by a single base learner and which usually causes underfitting (i.e., missing relevant relations between features and target outputs); (2) the variance, which arises from the base learner sensitivity to small fluctuations in the training set and causes overfitting (i.e., the algorithm models the noise present in the training data).

Ensemble approaches can be classified into two types, homogeneous ensembles and heterogeneous onss. Homogeneous ensembles consist of members having a single-type base learning algorithm, whereas heterogeneous ensembles consist of members having different base learning algorithms. The choice of base learners and their combination to build an ensemble is extremely important for building a successful model. To this end, several algorithms have been proposed in the literature [57]. These include both, simple aggregation strategies, such as majority voting, weighted majority voting, average voting, and more advanced ones like bagging, boosting, and stacking.

Majority Voting is a simple aggregation strategy that considers the prediction of each base classifier, for a new instance, as a single vote. It then assigns the class which obtains the largest number of votes to the new instance.

Weighted Majority Voting is an extension to Majority Voting strategy, except that it gives more weight to the best base classifier by counting its prediction twice. It then follows the same strategy as Majority Voting as it assigns, to the new instance, the class which obtains the largest number of votes.

Average Voting assigns to the new instance an averaged value of the predictions of all base classifiers.

In binary classification problems, such as defect prediction, where the output is either “defective” or “non-defective”, this strategy computes the average of the prediction probabilities yielded by each base classifier. If the final averaged probability is lower than 0.5, the instance is classified as non-defective, otherwise it is assigned to the defective class.

Stacking is an ensemble machine learning algorithm which uses a meta-classifier to learn how to best combine the predictions from two or more base machine learning algorithms. Stacking can harness the power of a range of well-performing models on a classification or regression task and can make predictions that achieve better performance than any single model in the ensemble.

Bagging and *Boosting* are considered homogeneous learners, while Stacking considers heterogeneous base learners. Besides, Bagging mainly focuses on producing an ensemble model with less variance than its components whereas Boosting and Stacking will mainly try to produce strong models less biased than their components.

2.3 Multi-Objective Evolutionary Optimisation

Evolutionary Algorithms (EAs) are evolution-based optimisation algorithms used to find approximation solutions in a feasible amount of time to otherwise hard search problems [32, 37]. EAs work by iteratively evolving a population of chromosomes (solutions), each of which containing a set of genes. These solutions are encoded in a pre-defined structure, also known as the *representation*, e.g., an array of bits, integers, or floating points. The quality of a solution is assessed by a *fitness function*, which measures the extent to which a solution is fit to solve the problem. At each generation, the solutions of the population (parents) undergo crossover and random mutations in order to generate new candidate solutions (offspring). These operations carry genetic information from the fittest parents to the offspring and introduce diversity into the population, respectively. At the end of the generation, the fittest solutions survive and become parents in the subsequent generation. When a given stopping condition is reached, the fittest solution is returned. However, in most real-world scenarios (such as in Software Engineering), there are many conflicting objectives with equal weights that should be considered when analysing the quality of solutions [37, 76, 77]. They are said to be *conflicting* because they often cannot be optimised simultaneously in full, i.e., by improving one objective, the others are likely to deteriorate. Therefore, Multi-Objective Evolutionary Algorithms (MOEAs) try to find a balance between the many objectives and eventually output a set of non-dominated solutions using the concept of Pareto dominance [37, 76] as explained below. Let Z be a set of minimisation objectives and x and y two different solutions. Solution x is said to dominate solution y $(x \succ y)$ if: $\forall z \in Z : z(x) \leq z(y) and \exists z \in Z : z(x) < z(y)$

If these conditions do not hold, then x and y are said to be non-dominated, i.e., they represent equally feasible solutions with acceptable trade-offs for the problem at hand.

3 MEG: MULTI-OBJECTIVE ENSEMBLE GENERATION

Figure 1 depicts an overview of MEG. The main purpose of MEG is to automatically generate ensemble classifiers by choosing a set of base classifiers, tuning them, and then selecting an aggregation strategy to produce the final ensemble. MEG is based on MOEAs, thus it iteratively evolves a population of ensembles across multiple generations and outputs the ensembles with the best trade-off between diversity of base classifiers and overall accuracy of predictions (Section 3.2 describes the objectives).

MEG differs from related work [17, 23, 25, 28, 59] in many ways. While algorithms like DIVACE [17] work as a Pareto-ensemble technique (evolve base classifiers and aggregate the non-dominated ones), MEG takes a more direct and intuitive approach where each solution in the population is a whole ensemble. Other work use a more similar representation as the one adopted by MEG [23, 25, 28, 59]. However, such approaches focus on selecting a set of pre-defined base classifiers, as opposed to designing, configuring, and building ensembles and the constituent parts.

MEG differs from related work by selecting base classifiers, optimising their hyper-parameters, and finally picking an ensemble strategy that best suits the context. The result is not a single ensemble, but rather a set of evolved ensembles from which the engineer can choose the one that best fits their needs. This allows for a robust evolution of ensembles with more flexibility in how they are designed and built. We define our proposed technique, MEG, as *whole ensemble generation*, which is not only novel for the defect prediction literature, but also for the general ensemble literature.

3.1 Representation

The representation of MEG consists of three arrays: i) Classifiers – binary array; ii) Parameters – double/floating points array; and iii) Ensemble/Aggregation Strategy – integer array.

The classifier array contains 15 bits, where each index corresponds to a specific classifier. If the bit in index i is 1, it signifies that the i-th classifier is active and will be included in the ensemble, otherwise, it will not be included. MEG explores three different types of classifiers. These are namely, Naive Bayes (NB) at indexes 1..3, three k-Nearest Neighbors algorithms (k-NN) at indexes 4..6, four Support Vector Machines (SVM) at indexes 7..10, and five Decision Trees (DT) at indexes 11..15. We included those models in the representation because the most related work [53] to ours used the same. This allows a fair comparison in our empirical study, however future work can extend MEG to incorporate other classifiers.

The parameter array consists of the hyper-parameters of each of the 15 classifiers.

For k-NN, SVM, and DT, the parameters are represented by double values indicating the number of neighbours, cache size, and pruning confidence, respectively. For NB, we use a categorical value where 0 indicates the normal density distribution and 1 represents a kernel density estimator. MEG can be extended to handle additional hyper-parameters for each classifier, however in this work we experimented with the ones used in the work of Petrić et al. [53].

Finally, the strategy array consists of a single integer value (later converted into a categorical one) representing the strategy to be used by the ensemble to aggregate the predictions of all constituent classifiers. Given that we aim at investigating ensembles composed by different types of base classifiers, we consider the following heterogeneous aggregation strategies: i) majority voting; ii) weighted majority voting; iii) stacking; and iv) average voting.[1]

The proposed representation allows MEG to simultaneously select classifiers, optimise their parameters, and select the aggregation strategy. For instance, differently from algorithms such as DIVACE [17], the strategy array allows MEG to optimise for a specific ensemble strategy, as opposed to forcing the engineer to choose one. Furthermore, allowing the selection of classifiers and their tuning online also reduces the engineering effort, as the engineer is not required to pre-train the classifiers before the optimisation process, such as in the work of Fletcher et al. [23].

3.2 Fitness Functions

MEG uses two fitness functions to guide the search for ensembles: diversity and accuracy. The accuracy of an ensemble depicts how

[1]We do not consider Bagging and Boosting as they are both homogeneous ensemble strategies which take into account a single type of base classifier.

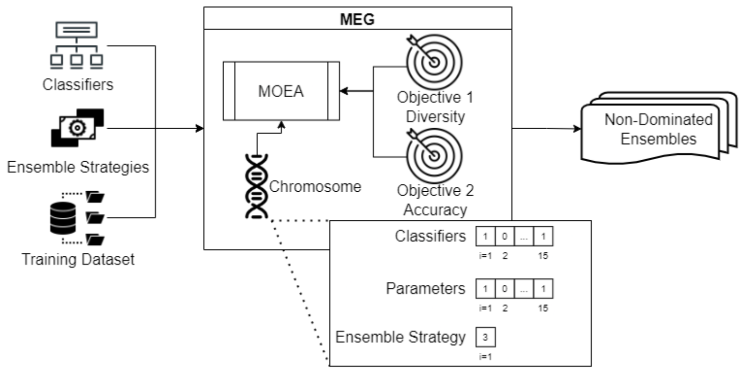

Figure 1: An overview of our proposed Multi-objective Ensemble Generation (MEG) approach.

well it can predict the labels of the instances under consideration, which in the context of this work are "defective" (true) or "non-defective" (false). Naturally, the more accurate the ensemble, the better. On the other hand, the diversity measures assess how different the predictions of the classifiers in the ensemble are. Diversity is an important factor when designing ensembles, since a diverse ensemble is more likely to predict "corner cases" instances. However, in a classification problem, the more the classifiers disagree in their predictions, the less accurate the ensemble tends to be. For instance, for a given instance with the true label "defective", if two classifiers each predict "defective" and "non-defective" respectively, then we obtain diverse results, but with 50% accuracy. Hence, these two measures are conflicting, but MEG still aims at optimising both for a good trade-off.

There are many accuracy and diversity measures in the literature [57]. In this work, we use Mathews Correlation Coefficient (MCC) [63] as the accuracy objective, and Disagreement [38] as the diversity measure. MCC represents the correlation coefficient between the actual and predicted classifications. Equation 1 shows the formula for the assessment of MCC.

$$\uparrow Accuracy = MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{1}$$

where True Positives (TP) are defective modules correctly classified as defective; False Positives (FP) are non-defective modules falsely classified as defective; False Negatives (FN) are defective modules falsely classified as non-defective; True Negatives (TN) are non-defective modules correctly classified as non-defective (see Table 1).

MCC outputs a value between -1 and $+1$, where a value of $+1$ indicates a perfect prediction, a value of 0 signifies that the prediction is no better than random guessing, and -1 represents a completely miss-classified output. With that in mind, the greater the MCC, the better the solution. We opted to use MCC to assess and compare the accuracy of models as this measure has been strongly recommend in alternative to other previously popular measures, such as F-measure, which have been shown to be biased [51, 63, 73] when the data is imbalanced (as it is frequently the case in DP). MCC is a more balanced measure which, unlike the other measures, takes into account all the values of the confusion matrix [51, 63].

Table 1: Confusion Matrix for Binary Classification.

Actual Value	Predicted Value	
	Defective (1)	Non-Defective (0)
Defective (1)	TP	FN
Non-Defective (0)	FP	TN

The second objective, diversity, is computed based on the disagreement measure [38], which measures (as the name implies) the prediction disagreement between groups of classifiers. We use this measure as it was used in previous work for traditional ensemble defect prediction [53] to which we compare MEG.

Equation 2 depicts the formula for computing Disagreement:

$$\uparrow Diversity = DIS_{i,j} = \frac{N^{10} + N^{01}}{N^{11} + N^{00} + N^{10} + N^{01}} \tag{2}$$

where N^{10} represents the number of instances correctly classified by the i^{th} classifier and incorrectly by the j^{th} classifier of the ensemble. N^{01}, N^{00} and N^{11} can be interpreted similarly. In summary, disagreement measures how many of the base classifiers' predictions contradict each other. Disagreement values can be any value within the range $[0, 1]$, where the higher the value, the more different the two classifiers. To calculate the diversity of the ensemble, we find the average pairwise disagreement between all pairs of its constituent members. An ideal ensemble would be the one that yields a high MCC and a high disagreement between its constituent classifiers. However, as mentioned at the beginning of this section, this is hard to achieve in practice, thus the need of using a MOEA to strike an optimal trade-off between these two competing goals.

3.3 Genetic Operators

Since there are three arrays in the representation (Section 3.1), the crossover and mutation happen in three parts, each of which with the respective operators adapted for the type of genes.

MEG uses a Single Point Crossover operator with 95% probability for all three representation arrays. This crossover operator takes two parents and combines their genes to generate two children. First, it chooses an index from the parents' genes array at random. It cuts the parents on that index into two parts, left and right. It then combines the left part of parent one with the right part of parent two to generate the first child, and the right genes of parent one with the left genes of parent two to create the second child.

After crossover, the children undergo mutation with a lower probability. Since the classifiers array (bit array) and parameters array (double array) have 15 indexes, the mutation probability is set to 0.07. Thus, it is expected that each child will have one of its bit/double gene mutated. For the ensemble strategy (int array) with one index, the probability is set to 0.25. Hence, it is expected that the ensemble aggregation strategy is mutated once in every four children. MEG uses a Bit Flip Mutation operator for the classifier array, and a Simple Random Mutation operator for the parameter and strategy arrays. The former simply flips the bit by changing it to 1 if the gene is 0, or to 0 if the gene is 1. The latter generates a random number for a mutated gene.

We set the population size to 100 and the stopping condition to 10,000 fitness evaluations. This means that the evolutionary process runs for 100 generations. In the end, all the non-dominated solutions are returned. In our experiments, the same algorithm, operators, and configuration are used with DIVACE (Sections 4 and 5).

3.4 Implementation Aspects

MEG was implemented using jMetal 5.10,[2] a widely-used framework for realising MOEAs in Java. jMetal provides multiple MOEAs, and mutation and crossover operators, which suits well the purpose of MEG. We implement MEG by using the NSGA-II [19] algorithm. We chose NSGA-II due to its popularity, easily adaptable nature with jMetal, performance, and because it has shown to be a robust algorithm in the SBSE literature [31]. For the ML implementation, we use the Weka 3.9 framework.[3] Our source-code is publicly available online, along with a replication package, containing the datasets, the raw results and the scripts we realised to analyse them [49].

4 EXPERIMENTAL DESIGN

In order to validate the effectiveness of MEG, we benchmark it against base classifiers, the state-of-the-art ensemble stacking for defect prediction as proposed by Petrić et al. [53], and the state-of-the-art Pareto-ensemble generation approach DIVACE [17], in the context of Cross-Version Defect Prediction (CVDP). Specifically, we aim at answering three research questions (RQs) as detailed below.

4.1 Research Questions

ML base classifiers have been widely proposed and considered as benchmarks in previous defect prediction studies. Therefore, our first research question investigates and compares the performance of MEG with that of traditional ML base classifiers. We consider this a "sanity check" given that any newly proposed model that cannot generally outperform base classifiers cannot be considered a scientific advancement in the state-of-the-art. To this end we ask:

RQ1 – MEG Vs. Base Classifiers: How does MEG compare to traditional base classifiers?

If MEG passes this sanity check, then we investigate whether our proposed approach can also outperform the aggregation of multiple ML base classifiers. We therefore compare the performance of MEG with that of existing ensemble for defect prediction. In order to verify this, we pose our second research question:

[2]https://github.com/jMetal/jMetal
[3]https://www.cs.waikato.ac.nz/ml/weka/

RQ2 – MEG Vs. Traditional Ensemble: How does MEG compare to ensemble stacking?

In particular, as nobody has previously proposed the use of MOEAs to generate ensemble models for defect prediction, we answer the above question by verifying whether MEG actually performs better than stacking, which has proven to be better than other ensemble approaches in previous defect prediction work [53].

Since our approach, MEG, is also novel with respect to existing multi-objective evolutionary approaches designed to generate ensemble for general-purpose classification problems, we also benchmark it with the state-of-the-art Pareto-ensemble generation technique DIVACE, thereby motivating our last research question:

RQ3 – MEG Vs. Pareto-ensemble: How does MEG compare to Pareto-ensemble generation, more specifically DIVACE, the state-of-the-art ensemble generation technique based on MOEAs?

In the following subsections we describe the techniques and datasets used as a benchmark to answer RQs 1–3, and the validation and evaluation criteria we employed to assess the performance of the prediction models we compare.

4.2 Benchmark Techniques

4.2.1 Base Classifiers. As a baseline benchmark, we compare the ensemble produced by MEG to the single based learners, as the purpose to build an ensemble is to achieve prediction performance which are at at least comparable or superior to individual classifiers [9, 13, 53]. We use the same four base classifiers used by Petrić et al. [53]., namely Naïve Bayes (NB)[70], k-Nearest Neighbour (k-NN), Support Vector Machine (SVM) [14], and Decision Tree (DT) [58]. To allow for a fair comparison with the ensembles, these are the same classifiers used by both MEG and DIVACE.

Based on the study by Petrić et al. [53], we train the base classifiers with the following parameters and select the configuration with the best training/validation MCC, as explained in Section 4.4: i) NB – NB with conventional distribution estimation and NB-K with kernel density estimator; ii) DT – pruning confidence values of 0.25, 0.2, 0.15, 0.1, and 0.05; iii) k-NN – number of neighbours (k) of 3, 5, and 7; and iv) SVM – C of 1, 10, 25, and 50.

4.2.2 Stacking Ensemble. The Stacking ensemble proposed by Petrić et al. [53] is composed by at most 15 base classifiers, each of which has a pre-defined configuration. The best stacking composition is the best combination of 2 to 15 base classifiers with the best training/validation MCC value. Moreover, since the other algorithms and MEG also use MCC, this was the natural choice for a fair comparison. To find the best Stacking combination, we followed the procedure proposed by Petrić et al. [53]. We first train the 15 base classifiers and order them in descending order according to their MCC. We then build 14 combinations from sizes 2 to 15, incrementally adding the next best base classifier, and setting the best classifier as the meta-classifier. We then select the best combination for a given program obtained on the train/validation set and use it during the testing phase, as detailed in Section 4.

4.2.3 Pareto-ensemble Generation: DIVACE. DIVACE is a Multi-objective Genetic Algorithm that uses the Pareto-ensemble approach to search for optimal ensemble classifiers, i.e., it joins the resulting non-dominated predictors into one single ensemble. DIVACE had never been assessed for defect predition in prior studies. It was originally proposed for a regression task (more specifically for building neural networks), but it has served as a reference since then for other ensemble generation techniques. It makes use of specialised continuous operators to generate weights and drive solutions towards specific and relevant parts of the search space. For more details on the approach we refer the reader to the original paper by Chandra et al. [17].

DIVACE, as originally proposed, uses the Negative Correlation Learning (NCL) measure for diversity, and MCC for the accuracy objective. DIVACE evolves the same four base classifiers as MEG by using NSGA-II, to allow for a fair comparison. Since DIVACE does not automatically select the aggregation strategy during the evolutionary process, one has to manually experiment with different aggregation strategies to identify the most suitable for the problem at hand. We therefore tried different strategies (i.e., Majority Voting, Stacking, and Average Rule) and found that Majority Voting generally generates the best results for the investigated datasets.

At the end of the evolution process, DIVACE aggregates all non-dominated solutions based on NCL and MCC into a single Majority Voting ensemble, thus producing a single optimal solution.

4.3 Datasets

In our empirical study we used the corpus made publicly available by Yatish et al. [74]. This data has been collected using a realistic approach based on two main criteria: (i) the use of an Issue Tracking System with the availability of high numbers of closed or fixed issue reports; and (ii) issue reports collected for each studied system are traceable back to the code. Following this procedure has resulted in obtaining less erroneous defect counts and hence representing a more realistic scenario of defective module collection.

We experiment with eight software systems considering three releases for each system, as listed in Table 2, which has been shown to be preferable to other types of validation such as 10-fold cross validation or bootstrapping [22, 29].

4.4 Validation Criteria

In the CVDP scenario, for each of the software systems, we train on one release and test on a later version, i.e., we train on version v_x and test on version v_y, where $x < y$, as done in previous work [29].

Training and Validation: For all algorithms included in this study, to prevent overfitting during the training phase, we apply an internal bootstrapping procedure with replacement using 80% of the data for training and 20% for validation (as suggested by Tantithamthavorn et al. [66]). We use this bootstrapping procedure to cater for the randomness of the data as confounding factors for the results. During the MOEA evolution, each candidate solution's fitness is the result of the predictions over the unseen 20% validation. We perform the same procedure to train the Stacking (state-of-the-art ensemble), DIVACE, and all traditional classifiers.

For each of the approaches investigated herein, the best performing solutions, are selected based on MCC on the validation data, and then used to train the final model on the whole training data. Such a model is then evaluated on an unseen test set. Given that MEG generates a set of ensembles (rather than a single solution as done by the base classifiers and DIVACE), we choose the solution in the Pareto-front with the highest MCC value on the validation data as the MEG's final solution. After a preliminary but comprehensive assessment, such a heuristic also showed to be the best one among other investigated ones such as selecting the most diverse solution, the one in between, or a random solution. We provide a summary of this preliminary assessment in our online artefact.

Testing: For each of the systems and prediction techniques considered herein, we build two prediction models. One by using the first available version in Yatish's corpus [74] as training data, given that recent work has proven that it is effective to use early defect data for training purposes [8, 65], and one by using the penultimate available version, as done in most of previous CVDP work. Both models are then tested on the latest version available in the corpus, which is completely unseen/untouched during the training process. By following this procedure, we ensure that the training and choice of best solution in the previous phases reflects a real-world scenario where the engineer does not possess information about the software components for which they are trying to predict defects. Moreover, the versions we used as train and test sets are not immediately subsequent releases nor are they the system's most recent ones. In addition, there is always a window of at least five months between these releases. This reduces the likelihood of the snoring effect or unrealistic labelling as described in previous studies [4, 8, 34].

Finally, to mitigate for the variability induced by the use of stochastic algorithms, we run the above procedure 30 times [7, 55].

In order to ensure a fair comparison among all algorithms, we ensured that the bootstrapping procedure samples the same data to train each compared approach within a same run by using a same seed for all, yet different runs uses different seeds.

Table 2: Total number of modules and percentage of faulty components for each of the datasets used in our empirical study. We used the two lowest version numbers (one at time) for training the prediction models and the highest one for testing them.

Dataset	No. of modules (faulty %)	Dataset	No. of modules (faulty %)
activemq-5.0.0	1884 (15.55%)	hive-0.9.0	1560 (11.28%)
activemq-5.3.0	2367 (10.90%)	hive-0.10.0	1560 (11.28%)
activemq-5.8.0	3420 (6.02%)	hive-0.12.0	2662 (8.00 %)
derby-10.2.1.6	1963 (33.67%)	jruby-1.1.0	731 (11.9%)
derby-10.3.1.4	2206 (30.33%)	jruby-1.5.0	1131 (7.25%)
derby-10.5.1.1	2705 (14.16%)	jruby-1.7.0	1614 (5.39%)
groovy-1.5.7	757 (3.43%)	lucene-2.3	805 (24.35%)
groovy-1.6-B1	821 (8.53%)	lucene-3.0	1337 (11.59%)
groovy-1.6-B2	884 (8.60%)	lucene-3.1	2806 (3.81%)
hbase-0.94.0	1059 (20.59%)	wicket-1.3.0-B1	1763 (7.37%)
hbase-0.95.0	1669 (22.95%)	wicket-1.3.0-B2	1763 (7.37%)
hbase-0.95.2	1834 (26.34%)	wicket-1.5.3	2578 (4.07%)

4.5 Evaluation Criteria

We use MCC to evaluate the prediction performance of the models given that we do not target a specific business context [45, 51], and, as explained in Section 3.2, MCC is a comprehensive measure, which provides a full picture of the confusion matrix by assessing all its aspects equally. It is also not sensitive to highly imbalanced data and is widely used in the defect prediction and machine learning literature [51, 63, 73].

In order to show whether there is any statistical significance between the results obtained by the models, we perform the Mann-Whitney U [42] setting the confidence limit, α, at 0.05 and applying the Bonferroni correction (α/K, where K is the number of hypotheses) when multiple hypotheses are tested. Unlike parametric tests, the Mann-Whitney U raises the bar for significance, by making no assumptions about underlying data distributions. Moreover, we used effect size to assess whether the statistical significance has practical significance effect size [7]. To this end we use the Vargha and Delaney's $\hat{A}_{12}$ non-parametric effect size measure, as it is recommended to use a standardised measure rather than a pooled one like the Cohen's d when not all samples are normally distributed [7], as in our case. The $\hat{A}_{12}$ statistic measures the probability that an algorithm A yields greater values for a given performance measure M than another algorithm B, based on the following equation:

$$\hat{A}_{12} = (R_1/m - (m+1)/2)/n \qquad (3)$$

where R_1 is the rank sum of the first data group we are comparing, and m and n are the number of observations in the first and second data sample, respectively. Values between $(0.44, 0.56)$ represent negligible differences, values between $[0.56, 0.64)$ and $(0.36, 0.44]$ represent small differences, values between $[0.64, 0.71)$ and $(0.29, 0:44]$ represent medium differences, and values between $[0.0, 0.29]$ and $[0.71, 1.0]$ represent large differences.

4.6 Threats to Validity

Threats to External Validity: As it is the case for most software engineering empirical studies, the datasets (i.e., programs) and prediction approaches used in this paper might not be fully representative of the population, which is a threat to the generalisation of our conclusions. The datasets we use are all based on open-source Java projects [74], which limits the generalisability of our results to proprietary software or projects written in other languages. In order to mitigate this threat, we used a variety of programs of different sizes and imbalanced nature, and which have been used in previous work [74]. Another external threat is linked to the choice of techniques which MEG was benchmarked against. We benchmarked MEG against the most relevant related work in the literature: traditional base ML classifiers, widely used in defect prediction work [27], the only multi-objective ensemble proposed thus far for defect prediction [53] and the state-of-the-art evolutionary ensemble DIVACE. Moreover, we implemented both DIVACE and the Stacker approach with Weka which is the tool used by Petric et al. [53] to closely reproduce the state-of-the art multi-objective ensemble DP [53]. This allowed us to share as many aspects as possible between all approaches and reduce any confounding factors that could arise from different implementations or configurations [40, 50]. While we believe that it would be interesting to carry out a large-scale empirical study comparing other ensemble approaches which have been proposed in the more general ML literature but have not been used for defect prediction (including for example auto-sklearn and AutoFolio [35]), this is out of the scope of our work and it deserves an investigation on its own right due to both the different aim, scope and technical challenges involved. For example, in order to compare various ensembles provided in ML tools other than Weka, such as auto-sklearn or AutoFolio which are currently available only in Phyton, it might require one to re-implement all approaches in a same tool to ensure a fair empirical comparison as using different ML tools might lead to different results [40, 50].

Threats to Internal Validity: The most prominent threat to internal validity relates to the correctness of our own implementations of DIVACE and the Stacking build procedure. We followed all the details provided in the reference papers [17, 53], but it is still possible that some differences were introduced. In order to mitigate this threat, all authors made sure to rule out any ambiguity by verifying the code and comparing it to the reference papers. In occasions where we could not agree on the way to resolve an ambiguity, we opted for design decisions which allow us to compare all the approaches on a level playing field.

Threats to Construct Validity: DIVACE and MEG are stochastic by nature. Consequently, the results of these techniques may differ from one run to another, causing unwanted variations in our experimental analysis. In order to mitigate this variability, and to provide a robust analysis, both DIVACE and MEG were executed 30 times and the median MCC values were reported (as opposed to means given that the latter is known to be more susceptible to outliers [64]), as suggested by best practices for the assessment of randomised optimisation algorithms and prediction systems [7, 51, 55, 64]. Moreover, we perform bootstrapping during training by using a same random seed for all techniques used across all our experiments. This ensures that any variation or difference in results is due to the nature of the techniques themselves and not due to a different random data sampling. Finally, we used a robust and unbiased measure, such as MCC, to evaluate the prediction capabilities of all the approaches investigated herein, and performed statistical tests, including both hypothesis testing and effect size, by carefully checking all the required assumption, such that our conclusions could be backed up by scientifically sound evidence [51].

5 RESULTS

In this section, we present and discuss the results of all research questions addressed in our work.

5.1 Answer to RQ1 – MEG Vs. Base Classifiers

As a sanity check, we compare the performance of MEG to that of traditional classifiers (i.e., DT, NB, k-NN and SVM) known to perform well for the task of defect prediction.

Table 3 shows that, out of the four classifiers investigated, NB achieves the best performance overall. When compared to MEG, results show that MEG performs similarly or better than NB in 69% of the cases with 73% of those cases being statistically significant and having a large effect size.

Our results also show that MEG is able to generate ensembles that perform similarly or better than traditional classifiers in 45 (70%) out of the 64 cases investigated. Moreover, MEG strictly outperforms traditional classifiers in 39 out of the 64 (61%) cases considered with the difference in 90% of those cases being statistically significant.

Answer to RQ1: MEG generates statistically better or at least equivalent ensembles to traditional classifiers in 70% of the cases.

Table 3: MCC values obtained on the test set by the base learners (NB, DT, k-NN, SVM), MEG, DIVACE, over 30 runs.

Version (training data)	MEG	DIVACE	Stacking	NB	DT	k-NN	SVM
activemq-5.0.0	0.29	0.20	0.24	0.29	0.15	0.25	0.30
derby-10.2.1.6	0.40	0.25	0.22	0.37	0.24	0.16	-0.05
groovy-1.5.7	0.23	0.31	0.22	0.21	0.31	0.33	0.24
hbase-0.94.0	0.30	0.27	0.25	0.07	0.27	0.25	0.31
hive-0.9.0	0.20	0.19	0.10	0.21	0.19	0.07	0.22
jruby-1.1	0.23	0.28	0.16	0.22	0.30	0.29	0.20
lucene-2.3.0	0.18	0.13	0.12	0.18	0.13	0.11	0.12
wicket-1.3.0-B1	0.08	0.20	0.13	0.18	0.20	0.16	0.15
activemq-5.3.0	0.31	0.27	0.27	0.28	0.24	0.23	0.29
derby-10.3.1.4	0.39	0.31	0.31	0.37	0.30	0.28	0.38
groovy-1.6-B1	0.20	0.26	0.27	0.21	0.26	0.47	0.26
hbase-0.95.0	0.30	0.18	0.19	0.33	0.26	0.20	0.31
hive-0.10.0	0.28	0.27	0.27	0.18	0.26	0.19	0.23
jruby-1.5.0	0.19	0.28	0.10	0.17	0.28	0.27	0.17
lucene-3.0.0	0.06	0.05	0.09	0.19	0.04	0.14	0.00
wicket-1.3.0-B2	0.15	0.00	0.00	0.19	0.10	0.11	0.12

5.2 Answer to RQ2 – MEG Vs. Traditional Ensemble

When compared to the state-of-the-art Stacking ensemble, results show that MEG is able to generate similarly or better performing models in 14 out of the 16 cases under study (88%) with the difference in 86% of those cases (12 out of 14) being statistically significant with a large effect size.

To understand the performance of the ensemble, we analyse the non-dominated solutions generated by MEG. We discovered that Stacking, as an aggregation strategy, was selected in only 3% of the cases; the least common type of generated ensembles. On the other hand, we found that the most selected ensemble strategy by MEG was Weighted Majority Voting, which simply counts the best classifier's vote twice and the other classifiers once. This strategy was selected in 78% of the non-dominated ensembles.

Answer to RQ2: MEG is able to generate ensembles that perform similarly or better than Stacking in 88% of the cases. The difference in results obtained by MEG is statistically significantly better in 75% of the cases with a large effect size.

5.3 Answer to RQ3 – MEG Vs. Pareto-ensemble

We also compare the ensembles generated by MEG to those generated by DIVACE. Results show that MEG outperforms DIVACE in 11 out of 16 cases studied (69%). This finding is also supported by the statistical tests showing that, in 82% of these cases, the difference is statistically significantly better and the effect size is large.

We further investigated the solutions generated by both these approaches and we found some interesting behavioural differences when it comes to the nature of the solutions. While MEG produced solutions consisting of a more heterogeneous set of classifiers with a mean and median number of classifiers equal to 2.41 and 2, respectively, DIVACE's behaviour was more homogeneous. In most cases, DIVACE generated non-dominated solutions that were retained across generations resulting in ensembles comprised of 100 classifiers, out of which ≈98 were entirely of a homogeneous nature. MEG, on the other hand, generated a smaller set of heterogeneous ensembles with the largest number of classifiers being equal to 11. This shows that an ensemble consisting of a small heterogeneous set of classifiers yields better results than one comprising of a large set of homogeneous classifiers.

Answer to RQ3: MEG generates ensembles that are similar or statistically better to those produced by DIVACE in 69% of the cases, with the difference in 82% of them being statistically significant.

5.4 Final Remarks

MEG yielded statistically significantly better results than the other algorithms in most of the cases. To be precise, if we consider all comparisons between MEG and the other techniques, MEG is able to generate ensembles producing similar or more accurate predictions than those produced by the other approaches in 73% of the cases (with favourable large effect sizes in 80% of them). In the minority of the cases (27%) where MEG does not statistically significantly outperform all other approaches we observed that the number of faulty modules in the training data is lower than 10%, or the training data consists of less than 1,000 modules. For these cases, we also observed that MEG's ensembles achieve better MCC values than those produced by traditional ML and Pareto-ensemble in the training phase, thus suggesting that its ensembles may overfit when the training data is small or contain few of defective instance.

The positive results obtained by using MEG in the majority of the cases investigated herein, suggest its use could be convenient for several reasons.

First, MEG removes from the engineer hands the burdensome, error-prone, and time-consuming task of building, or even selecting, an ensemble/classifier. Since it is all automated, the engineer can simply use MEG to generate robust ensembles.

Table 4: Mann-Withney U pair-wise test results / Vargha-Delaney $\hat{A}_{12}$ effect sizes obtained comparing MEG with DIVACE, Stacking, and base classifiers (NB, DT, k-NN, SVM). $\hat{A}_{12}$: Large – L; Medium – M; Small – S; Negligible – N. Cells highlighted in blue (p-value < 0.05 and effect size > 0.5) indicate that MEG **is statistically significantly better than the algorithms in the corresponding columns. Cells highlighted in orange (p-value < 0.05 and effect sizes < 0.5) indicate that** MEG **is significantly statistically worse. The last three rows show the number of times MEG yields better, equivalent, and worse results, respectively.**

Version (training data)	DIVACE	Stacking	NB	DT	k-NN	SVM
activemq-5.0.0	<0.01 / 1.0 (L)	<0.01 / 1.0 (L)	<0.01 / 1.0 (L)	<0.01 / 1.0 (L)	<0.01 / 1.0 (L)	<0.01 / 0.0 (L)
derby-10.2.1.6	<0.01 / 0.8 (L)	<0.01 / 0.83 (L)	<0.01 / 0.8 (L)	<0.01 / 0.83 (L)	<0.01 / 0.83 (L)	<0.01 / 1.0 (L)
groovy-1.5.7	<0.01 / 0.14 (L)	<0.01 / 1.0 (L)	<0.01 / 1.0 (L)	<0.01 / 0.0 (L)	<0.01 / 0.0 (L)	<0.01 / 0.03 (L)
hbase-0.94.0	<0.01 / 0.93 (L)	<0.01 / 0.93 (L)	<0.01 / 1.0 (L)	<0.01 / 0.93 (L)	<0.01 / 0.93 (L)	0.058 / 0.37 (S)
hive-0.9.0	0.197 / 0.6 (S)	<0.01 / 1.0 (L)	<0.01 / 0.0 (L)	1.0 / 0.5 (N)	<0.01 / 1.0 (L)	<0.01 / 0.0 (L)
jruby-1.1	<0.01 / 0.24 (L)	0.638 / 0.53 (N)	0.638 / 0.53 (N)	<0.01 / 0.03 (L)	<0.01 / 0.03 (L)	0.638 / 0.53 (N)
lucene-2.3.0	<0.01 / 1.0 (L)	<0.01 / 1.0 (L)	<0.01 / 0.83 (L)	<0.01 / 1.0 (L)	<0.01 / 1.0 (L)	<0.01 / 1.0 (L)
wicket-1.3.0-B1	<0.01 / 0.25 (L)	0.64 / 0.47 (N)	0.345 / 0.43 (N)	<0.01 / 0.0 (L)	0.64 / 0.47 (N)	0.64 / 0.47 (N)
activemq-5.3.0	<0.01 / 0.83 (L)	<0.01 / 1.0 (L)	0.151 / 0.6 (S)	<0.01 / 1.0 (L)	<0.01 / 1.0 (L)	0.151 / 0.6 (S)
derby-10.3.1.4	<0.01 / 1.0 (L)	<0.01 / 1.0 (L)	<0.01 / 0.77 (L)	<0.01 / 1.0 (L)	<0.01 / 1.0 (L)	<0.01 / 0.77 (L)
groovy-1.6-B1	<0.01 / 0.0 (L)	<0.01 / 0.0 (L)	<0.01 / 0.0 (L)	<0.01 / 0.0 (L)	<0.01 / 0.0 (L)	<0.01 / 0.0 (L)
hbase-0.95.0	<0.01 / 0.96 (L)	<0.01 / 1.0 (L)	<0.01 / 0.0 (L)	<0.01 / 1.0 (L)	<0.01 / 1.0 (L)	0.057 / 0.37 (S)
hive-0.10.0	<0.01 / 0.92 (L)	<0.01 / 0.9 (L)	<0.01 / 1.0 (L)	<0.01 / 1.0 (L)	<0.01 / 1.0 (L)	<0.01 / 1.0 (L)
jruby-1.5.0	<0.01 / 0.12 (L)	<0.01 / 1.0 (L)	<0.01 / 0.9 (L)	<0.01 / 0.0 (L)	<0.01 / 0.0 (L)	<0.01 / 0.9 (L)
lucene-3.0.0	0.587 / 0.54 (N)	<0.01 / 0.07 (L)	<0.01 / 0.0 (L)	<0.01 / 0.8 (L)	<0.01 / 0.0 (L)	<0.01 / 1.0 (L)
wicket-1.3.0-B2	<0.01 / 0.99 (L)	<0.01 / 1.0 (L)	<0.01 / 0.1 (L)	<0.01 / 0.93 (L)	<0.01 / 0.93 (L)	<0.01 / 0.93 (L)
MEG is better	9	12	8	10	10	7
MEG is equivalent	2	2	3	1	1	5
MEG is worse	5	2	5	5	5	4

Second, MEG can generate ensembles that can yield good predictions for future versions of a software. This is shown by the results achieved by the generated ensembles when trained on the first available software version, and tested on the latest.

Third, MEG can be extended and adapted to the engineers' needs. In this paper, we focused on MCC as a measure of performance accuracy given that it is a comprehensive and balanced measure [51]. However, there might be cases where it may be more important to minimize one type of classification error. For example, an engineer might want to reduce the number of false negatives to the very minimum when predicting defects for mission and safety critical software systems (e.g., software controlling autopilot, medical devices) in which even a single failure can have serious adverse effects [51]. In this case, MEG can be extended to be guided by a fitness function devised for a specific goal.

Regarding the qualitative results, we analysed possible correlations between the results of the generated ensembles and possible imbalance in the data with respect to the percentages of defective and non-defective classes. However, we did not find any clear pattern. It seems that the results of the ensembles generated by MEG are more influenced by the individual predictions of each base classifier. For instance, when predicting defects for *Derby*, SVM showed very good results during training (training and validation MCC values higher than 0.7), but very poor performance during testing (negative MCC value). When analysing the ensembles generated by MEG for this program, we found out that when SVM was added to the ensemble, the resulting MCC values would drop from around 0.40 to 0.02. The drop is noticeable and undeniably significant, but luckily it is an exception, not the rule.

In our experiments, each of MEG's independent runs took from 6–48 hours, which is greater than the time needed to perform the automated hyper-parameter tuning of the state-of-the-art algorithms (2–24 hours). While the time taken by the state-of-the-art is generally lower as it includes only training and possibly tuning, MEG's running time is higher because it also encompasses the automatic selection of classifiers, voting strategies, and tuning of parameters. This procedure is analogous to the experimentation, tuning, and selection of many different algorithms that are usually done manually by the engineers. If we compare the time and expertise that would be required by the engineer to perform these tasks, using MEG saves time and effort overall. This is a common trade-off for automated search based approaches, as observed in previous work [26].

6 RELATED WORK

Search-based Software Engineering has been shown to be a powerful tool to address Software Engineering prediction tasks [60], such as software effort estimation, change prediction, defect prediction and maintainability prediction [41]. In the context of defect prediction previous studies have investigated the use of both single- (e.g., [30, 48, 62, 68]) and multi-objective search-based approaches [15] to either build or fine-tune learning models. However, no previous defect prediction study has investigated the use of search-based approaches to guide the construction of ensemble models, which have instead been exploited to solve general-purpose classification tasks (see Section 6.2). On the other hand, several ensemble learning techniques have been previously explored to build defect prediction models [5, 6, 39, 43, 56, 71], however to the best of our knowledge, the work by Petrić et al. [53] is the only to contemplate diversity, together with accuracy, in order to build more robust ensembles.

We refer the reader to existing literature surveys for more details on machine learning models for defect prediction [27], search-based approaches for defect prediction [41], ensemble models for defect prediction [2, 43], and general-purpose evolutionary ensembles [57]. The rest of this section focuses on the work most related to ours.

6.1 Defect Prediction Ensemble Based on Diversity and Accuracy

Bowes et al. [11] and Panichella et al. [52] were the first to investigate the role of diversity in machine learning classifiers for defect prediction, unveiling that different classifiers predict different subsets of defects in datasets [11, 12, 52]. Subsequently, Petrić et al. investigated the use of diversity as a measure, in combination with accuracy, to guide the engineer in the design of ensembles for defect prediction [53] based on the intuition that the combination of an explicit diversity technique with a stacking ensemble could improve its prediction performance. To validate their proposal they carried out a thorough empirical study by comparing several stacking procedures manually designed to investigate how to best combine base classifiers based on diversity and accuracy measures, and they compared these with other types of ensembles such as bagging and boosting. They found that using a stacking built based to both, accuracy and diversity measures, produced the best results. We further exploit this intuition, and propose MEG which leverages the power of MOEAs to automatically search for optimal ensemble according to accuracy and diversity. Our proposal has the main advantage of relieving the engineer from the burden of manually designing an ensemble, which is instead required in previous work [53]. Besides, our results show that MEG outperforms the best ensemble defect prediction found by Petrić et al. [53] (see Section 5).

6.2 MOEAs for Ensemble Generation

Previous work has explored the use of Multi-Objective Evolutionary Algorithms (MOEAs) [37] in the context of ensemble generation by considering both accuracy and diversity as target objectives for different domains [1, 17, 24, 46] but none had been previously employed in the context of defect prediction.

All these work rely on the Pareto approach to generate the ensembles (see Section 2.3). Among these, the closest work to ours is that by Chandra and Yao [17], who used multi-objective optimisation based on NSGA-II in their algorithm called DIVACE, to evolve and train the set of weights of 3-layer neural networks by simultaneously optimising accuracy and diversity.

Studies in the AutoML literature have also investigated EAs for such optimisations, such as Autostacker [18]. However, work in this area generally optimises the parameters of the classifier composing the multi-layered ensemble, but does not encompass their selection or the aggregation strategy choice. Other work used MOEAs to make an optimal selection of pre-defined (or pre-trained) classifiers with feature selection for other classification tasks [18, 23, 25, 28, 59]. None leverage diversity as an objective.

In this paper, we instead propose an alternative and novel way to automatically generate ensemble models, based on *whole-ensemble generation* which is guided by the simultaneous use of diversity and accuracy. MEG has several advantages (as explained in Sections 3 and 5.4) and has achieved similar or significantly better results than DIVACE in our empirical study (see Section 5).

7 CONCLUSIONS AND FUTURE WORK

In this paper, we proposed MEG, a Multi-objective Ensemble Generation algorithm for defect prediction.

MEG is a novel technique, which, unlike previous MOEA-based ensemble generation, evolves a population of ensembles (i.e., *whole-ensemble generations*) rather than a population of base classifiers (i.e, *Pareto-ensemble generation*). Furthermore, MEG relies on both accuracy and diversity of ensembles to guide the search towards robust ensembles, which has not been previously evaluated in the context of defect prediction.

In our large-scale empirical study involving a total of 24 software versions from 8 different open-source projects, and both baselines and state-of-the-art benchmarks, we show that MEG is able to generate ensembles that achieve comparable or statistically significantly more accurate predictions for 73% of the comparisons, with large effect sizes in 80% of these cases.

These results show that our novel *whole-ensemble generation* approach, MEG, not only advances the state-of-the-art of ensemble in defect prediction (where multi-objective evolutionary ensemble had never been investigated) but it is also more effective than *Pareto-ensemble generation* algorithm. In addition, the benefits of MEG do not only lie on the higher accuracy of the generated ensembles, but also on the benefits of having an approach that automatically searches for an optimal design: This spares the engineer from the tedious, time-consuming, and error-prone activity of manually designing and experimenting.

The proposal of a novel *whole-ensemble generation*, such as MEG, opens up a variety of avenues for future work, including but not limited to the following:

- Investigating the effectiveness of *alternative solution representations* considering additional learners and aggregation strategies.
- Investigating the *effect of using other measures as fitness function* to guide the ensemble evolution, in combination or in alternative to the measures explored herein. These include the use of other diversity measures, and performance measures such as the classification stability [67] and the effort required for source code inspection [33, 44].
- Investigating *MOEAs other than NSGA-II* (e.g., MOEA/D, IBEA, MOCell, SPEA2) in order to assess to what extent the effectiveness of MEG varies depending on the underlying multi-objective algorithm [69].
- Investigating *Deep-Learning (DL) in combination with* MEG, to assess if this would further increase the ensemble prediction performance. DL approaches have recently been proposed to extract from the source code features which are subsequently used as an input (together or in alternative to traditional code metrics) to train a base binary classifier [72] for defect prediction. Our proposed approach, MEG, uses traditional code metrics yet it combines multiple different binary classifiers automatically searching for the best configuration. These approaches are different in nature: DL augments the training data, while MEG searches for an optimal configuration of multiple binary classifiers. It would be interesting in future work to explore such a combined use.

ACKNOWLEDGMENTS

Rebecca Moussa, Giovani Guizzo and Federica Sarro are funded by the ERC advanced fellowship grant 741278 (EPIC: Evolutionary Program Improvement Collaborators).

REFERENCES

[1] Hussein A Abbass. 2003. Pareto neuro-evolution: Constructing ensemble of neural networks using multi-objective optimization. In *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.*, Vol. 3. IEEE, 2074–2080.

[2] Wasif Afzal and Richard Torkar. 2011. Review: On the Application of Genetic Programming for Software Engineering Predictive Modeling: A Systematic Review. *Expert Syst. Appl.* 38, 9 (sep 2011), 11984–11997. https://doi.org/10.1016/j.eswa.2011.03.041

[3] Wasif Afzal, Richard Torkar, Robert Feldt, and Tony Gorschek. 2014. Prediction of faults-slip-through in large software projects: an empirical evaluation. *Software quality journal* 22, 1 (2014), 51–86.

[4] Aalok Ahluwalia, Massimiliano Di Penta, and Davide Falessi. 2020. On the Need of Removing Last Releases of Data When Using or Validating Defect Prediction Models. *arXiv preprint arXiv:2003.14376* (2020).

[5] Hamad Alsawalqah, Neveen Hijazi, Mohammed Eshtay, Hossam Faris, Ahmed Al Radaideh, Ibrahim Aljarah, and Yazan Alshamaileh. 2020. Software Defect Prediction Using Heterogeneous Ensemble Classification Based on Segmented Patterns. *Applied Sciences* 10, 5 (2020). https://doi.org/10.3390/app10051745

[6] Arsalan Ahmed Ansari, Amaan Iqbal, and Bibhudatta Sahoo. 2020. Heterogeneous Defect Prediction Using Ensemble Learning Technique. In *Artificial Intelligence and Evolutionary Computations in Engineering Systems*, Subhransu Sekhar Dash, C. Lakshmi, Swagatam Das, and Bijaya Ketan Panigrahi (Eds.). Springer Singapore, Singapore, 283–293.

[7] Andrea Arcuri and Lionel Briand. 2014. A Hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Software Testing, Verification and Reliability* 24, 3 (May 2014), 219–250. https://doi.org/10.1002/stvr.1486

[8] Abdul Ali Bangash, Hareem Sahar, Abram Hindle, and Karim Ali. 2020. On the time-based conclusion stability of cross-project defect prediction models. *Empirical Software Engineering* 25, 6 (2020), 5047–5083.

[9] Yijun Bian and Huanhuan Chen. 2021. When Does Diversity Help Generalization in Classification Ensembles? *IEEE Transactions on Cybernetics* (2021), 1–17. https://doi.org/10.1109/tcyb.2021.3053165

[10] David Bowes, Tracy Hall, Mark Harman, Yue Jia, Federica Sarro, and Fan Wu. 2016. Mutation-aware fault prediction. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*. 330–341.

[11] David Bowes, Tracy Hall, and Jean Petrić. 2015. Different Classifiers Find Different Defects Although With Different Level of Consistency. In *Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering*. ACM. https://doi.org/10.1145/2810146.2810149

[12] David Bowes, Tracy Hall, and Jean Petrić. 2017. Software defect prediction: do different classifiers find the same defects? *Software Quality Journal* 26, 2 (Feb. 2017), 525–552. https://doi.org/10.1007/s11219-016-9353-3

[13] Gavin Brown, Jeremy Wyatt, Rachel Harris, and Xin Yao. 2005. Diversity creation methods: a survey and categorisation. *Information Fusion* 6, 1 (March 2005), 5–20. https://doi.org/10.1016/j.inffus.2004.04.004

[14] Christopher JC Burges. 1998. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery* 2, 2 (1998), 121–167.

[15] Gerardo Canfora, Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella. 2015. Defect prediction as a multiobjective optimization problem. *Software Testing, Verification and Reliability* 25, 4 (2015), 426–459.

[16] Cagatay Catal and Banu Diri. 2009. A systematic review of software fault prediction studies. *Expert systems with applications* 36, 4 (2009), 7346–7354.

[17] Arjun Chandra and Xin Yao. 2006. Ensemble Learning Using Multi-Objective Evolutionary Algorithms. *Journal of Mathematical Modelling and Algorithms* 5, 4 (March 2006), 417–445. https://doi.org/10.1007/s10852-005-9020-3

[18] Boyuan Chen, Harvey Wu, Warren Mo, Ishanu Chattopadhyay, and Hod Lipson. 2018. Autostacker: A Compositional Evolutionary Learning System. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Kyoto, Japan) *(GECCO'18)*. 402—-409. https://doi.org/10.1145/3205455.3205586

[19] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197. https://doi.org/10.1109/4235.996017

[20] Sergio Di Martino, Filomena Ferrucci, Carmine Gravino, and Federica Sarro. 2011. A genetic algorithm to configure support vector machines for predicting fault-prone components. In *International conference on product focused software process improvement*. Springer, 247–261.

[21] Marco D'Ambros, Michele Lanza, and Romain Robbes. 2012. Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empirical Software Engineering* 17, 4-5 (2012), 531–577.

[22] Davide Falessi, Jacky Huang, Likhita Narayana, Jennifer Fong Thai, and Burak Turhan. 2020. On the need of preserving order of data when validating within-project defect classifiers. *Empirical Software Engineering* (2020), 1–26.

[23] Sam Fletcher, Brijesh Verma, and Mengjie Zhang. 2020. A non-specialized ensemble classifier using multi-objective optimization. *Neurocomputing* 409 (Oct. 2020), 93–102. https://doi.org/10.1016/j.neucom.2020.05.029

[24] Christian Gagné, Michèle Sebag, Marc Schoenauer, and Marco Tomassini. 2007. Ensemble learning for free with evolutionary algorithms?. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation - GECCO '07*. ACM Press. https://doi.org/10.1145/1276958.1277317

[25] Shenkai Gu and Yaochu Jin. 2014. Generating diverse and accurate classifier ensembles using multi-objective optimization. In *2014 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making (MCDM)*. IEEE. https://doi.org/10.1109/mcdm.2014.7007182

[26] Giovani Guizzo, Federica Sarro, Jens Krinke, and Silvia Regina Vergilio. 2020. Sentinel: A Hyper-Heuristic for the Generation of Mutant Reduction Strategies. *IEEE Transactions on Software Engineering* (2020), 1–1. https://doi.org/10.1109/TSE.2020.3002496

[27] Tracy Hall, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. 2011. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering* 38, 6 (2011), 1276–1304.

[28] Kate Han, Tien Pham, Trung Hieu Vu, Truong Dang, John McCall, and Tien Thanh Nguyen. 2021. VEGAS: A Variable Length-Based Genetic Algorithm for Ensemble Selection in Deep Ensemble Learning. In *Intelligent Information and Database Systems*. Springer International Publishing, 168–180. https://doi.org/10.1007/978-3-030-73280-6_14

[29] Mark Harman, Syed Islam, Yue Jia, Leandro L Minku, Federica Sarro, and Komsan Srivisut. 2014. Less is more: Temporal fault predictive performance over multiple hadoop releases. In *International Symposium on Search Based Software Engineering*. Springer, 240–246.

[30] Mark Harman, Syed Islam, Yue Jia, Leandro L. Minku, Federica Sarro, and Komsan Srivisut. 2014. Less is More: Temporal Fault Predictive Performance over Multiple Hadoop Releases. In *Search-Based Software Engineering*. Springer International Publishing, Cham, 240–246.

[31] Mark Harman, Phil McMinn, Jerffeson Teixeira De Souza, and Shin Yoo. 2011. Search based software engineering: Techniques, taxonomy, tutorial. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7007 LNCS (2011), 1–59. https://doi.org/10.1007/978-3-642-25231-0_1

[32] John H. Holland. 1992. *Adaptation in Natural and Artificial Systems An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* MIT Press. 232 pages.

[33] Qiao Huang, Xin Xia, and David Lo. 2017. Supervised vs Unsupervised Models: A Holistic Look at Effort-Aware Just-in-Time Defect Prediction. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 159–170. https://doi.org/10.1109/ICSME.2017.51

[34] Matthieu Jimenez, Renaud Rwemalika, Mike Papadakis, Federica Sarro, Yves Le Traon, and Mark Harman. 2019. The importance of accounting for real-world labelling when predicting software vulnerabilities. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 695–705.

[35] Jasper Jolly. 2019. *Passenger anger as tens of thousands hit by BA systems failure.* https://www.theguardian.com/business/2019/aug/07/british-airways-it-glitch-causes-disruption-for-passengers-delays

[36] Sunghun Kim. 2012. Defect, defect, defect: Defect prediction 2.0. In *Proceedings of the 8th International Conference on Predictive Models in Software Engineering*. 1–2.

[37] Abdullah Konak, David W. Coit, and Alice E. Smith. 2006. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety* 91, 9 (Sept. 2006), 992–1007. https://doi.org/10.1016/j.ress.2005.11.018

[38] Ludmila I Kuncheva and Christopher J Whitaker. 2003. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine learning* 51, 2 (2003), 181–207.

[39] Issam H. Laradji, Mohammad Alshayeb, and Lahouari Ghouti. 2015. Software defect prediction using ensemble learning on selected features. *Information and Software Technology* 58 (2015), 388–402. https://doi.org/10.1016/j.infsof.2014.07.005

[40] Cynthia C. S. Liem and Annibale Panichella. 2020. Run, Forest, Run? On Randomization and Reproducibility in Predictive Software Engineering.

[41] Ruchika Malhotra, Megha Khanna, and Rajeev R. Raje. 2017. On the application of search-based techniques for software engineering predictive modeling: A systematic review and future directions. *Swarm and Evolutionary Computation* 32 (2017), 85–109. https://doi.org/10.1016/j.swevo.2016.10.002

[42] H. B. Mann and D. R. Whitney. 1947. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics* 18, 1 (1947), 50–60. https://doi.org/10.1214/aoms/1177730491

[43] Faseeha Matloob, Taher M Ghazal, Nasser Taleb, Shabib Aftab, Munir Ahmad, Muhammad Adnan Khan, Sagheer Abbas, and Tariq Rahim Soomro. 2021. Software Defect Prediction using Ensemble Learning: A Systematic Literature Review. *IEEE Access* (2021).

[44] Thilo Mende and Rainer Koschke. 2010. Effort-Aware Defect Prediction Models. In *2010 14th European Conference on Software Maintenance and Reengineering*. 107–116. https://doi.org/10.1109/CSMR.2010.18

[45] Tim Menzies, Alex Dekhtyar, Justin Distefano, and Jeremy Greenwald. 2007. Problems with Precision: A Response to "Comments on 'Data Mining Static Code Attributes to Learn Defect Predictors'". *IEEE Transactions on Software Engineering* 33, 9 (2007), 637–640. https://doi.org/10.1109/TSE.2007.70721

[46] Leandro L. Minku and Xin Yao. 2013. An analysis of multi-objective evolutionary algorithms for training ensemble models based on different performance measures in software effort estimation. In *Proceedings of the 9th International Conference on Predictive Models in Software Engineering*. ACM. https://doi.org/10.1145/2499393.2499396

[47] Raimund Moser, Witold Pedrycz, and Giancarlo Succi. 2008. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Proceedings of the 30th international conference on Software engineering*. 181–190.

[48] Rebecca Moussa and Danielle Azar. 2017. A PSO-GA approach targeting fault-prone software modules. *Journal of Systems and Software* 132 (2017), 41–49.

[49] Rebecca Moussa, Giovani Guizzo, and Federica Sarro. 2022. MEG - On-line GitHub repository. https://github.com/SOLAR-group/MEG

[50] Rebecca Moussa and Federica Sarro. 2022. Do Not Take It for Granted: Comparing Open-Source Libraries for Software Development Effort Estimation. https://arxiv.org/pdf/2207.01705.pdf

[51] Rebecca Moussa and Federica Sarro. 2022. On the Use of Evaluation Measures for Defect Prediction Studies. In *2022 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. ACM.

[52] Annibale Panichella, Rocco Oliveto, and Andrea De Lucia. 2014. Cross-project defect prediction models: L'union fait la force. In *2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*. IEEE, 164–173.

[53] Jean Petrić, David Bowes, Tracy Hall, Bruce Christianson, and Nathan Baddoo. 2016. Building an Ensemble for Software Defect Prediction Based on Diversity Selection. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM. https://doi.org/10.1145/2961111.2962610

[54] Foyzur Rahman and Premkumar Devanbu. 2013. How, and why, process metrics are better. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 432–441.

[55] Paul Ralph, Nauman bin Ali, Sebastian Baltes, Domenico Bianculli, Jessica Diaz, Yvonne Dittrich, Neil Ernst, Michael Felderer, Robert Feldt, Antonio Filieri, Breno Bernard Nicolau de França, Carlo Alberto Furia, Greg Gay, Nicolas Gold, Daniel Graziotin, Pinjia He, Rashina Hoda, Natalia Juristo, Barbara Kitchenham, Valentina Lenarduzzi, Jorge Martínez, Jorge Melegati, Daniel Mendez, Tim Menzies, Jefferson Molleri, Dietmar Pfahl, Romain Robbes, Daniel Russo, Nyyti Saarimäki, Federica Sarro, Davide Taibi, Janet Siegmund, Diomidis Spinellis, Miroslaw Staron, Klaas Stol, Margaret-Anne Storey, Davide Taibi, Damian Tamburri, Marco Torchiano, Christoph Treude, Burak Turhan, Xiaofeng Wang, and Sira Vegas. 2021. Empirical Standards for Software Engineering Research. arXiv:2010.03525 [cs.SE]

[56] Santosh Rathore and Sandeep Kumar. 2021. An empirical study of ensemble techniques for software fault prediction. *Applied Intelligence* 51 (June 2021), 1–30. https://doi.org/10.1007/s10489-020-01935-6

[57] Ye Ren, Le Zhang, and P.N. Suganthan. 2016. Ensemble Classification and Regression-Recent Developments, Applications and Future Directions [Review Article]. *IEEE Computational Intelligence Magazine* 11, 1 (2016), 41–53. https://doi.org/10.1109/MCI.2015.2471235

[58] Steven L Salzberg. 1994. C4. 5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993.

[59] E.M. Dos Santos, R. Sabourin, and P. Maupin. 2006. Single and Multi-Objective Genetic Algorithms for the Selection of Ensemble of Classifiers. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings*. IEEE. https://doi.org/10.1109/ijcnn.2006.247267

[60] Federica Sarro. 2019. Search-Based Predictive Modelling for Software Engineering: How Far Have We Gone?. In *Search-Based Software Engineering - 11th International Symposium, SSBSE 2019, Tallinn, Estonia, August 31 - September 1, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11664)*. Springer, 3–7. https://doi.org/10.1007/978-3-030-27455-9_1

[61] Federica Sarro, Sergio Di Martino, Filomena Ferrucci, and Carmine Gravino. 2012. A further analysis on the use of genetic algorithm to configure support vector machines for inter-release fault prediction. In *Proceedings of the 27th annual ACM symposium on applied computing*. 1215–1220.

[62] Federica Sarro, Sergio Di Martino, Filomena Ferrucci, and Carmine Gravino. 2012. A Further Analysis on the Use of Genetic Algorithm to Configure Support Vector Machines for Inter-Release Fault Prediction. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC '12)*. ACM, New York, NY, USA, 1215–1220. https://doi.org/10.1145/2245276.2231967

[63] Martin Shepperd, David Bowes, and Tracy Hall. 2014. Researcher bias: The use of machine learning in software defect prediction. *IEEE Transactions on Software Engineering* 40, 6 (2014), 603–616.

[64] Martin Shepperd and Steve MacDonell. 2012. Evaluating prediction systems in software project estimation. *Information and Software Technology* 54, 8 (2012), 820–827.

[65] NC Shrikanth, Suvodeep Majumder, and Tim Menzies. 2021. Early Life Cycle Software Defect Prediction. Why? How?. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 448–459.

[66] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. 2016. An empirical comparison of model validation techniques for defect prediction models. *IEEE Transactions on Software Engineering* 43, 1 (2016), 1–18.

[67] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. 2017. An Empirical Comparison of Model Validation Techniques for Defect Prediction Models. *IEEE Transactions on Software Engineering* 43, 1 (2017), 1–18. https://doi.org/10.1109/TSE.2016.2584050

[68] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. 2019. The Impact of Automated Parameter Optimization on Defect Prediction Models. *IEEE Transactions on Software Engineering* 45, 7 (2019), 683–711. https://doi.org/10.1109/TSE.2018.2794977

[69] Vali Tawosi, Federica Sarro, Alessio Petrozziello, and Mark Harman. 2021. Multi-Objective Software Effort Estimation: A Replication Study. *IEEE Transactions on Software Engineering* (2021), 1–1. https://doi.org/10.1109/TSE.2021.3083360

[70] Ian H Witten, Eibe Frank, and Mark A Hall. 2005. Practical machine learning tools and techniques. *Morgan Kaufmann* (2005), 578.

[71] Xiaoxing Yang, Xin Li, Wushao Wen, and Jianmin Su. 2019. An Investigation of Ensemble Approaches to Cross-Version Defect Prediction. 437–442. https://doi.org/10.18293/SEKE2019-113

[72] Xinli Yang, David Lo, Xin Xia, Yun Zhang, and Jianling Sun. 2015. Deep learning for just-in-time defect prediction. In *2015 IEEE International Conference on Software Quality, Reliability and Security*. IEEE, 17–26.

[73] Jingxiu Yao and Martin Shepperd. 2020. Assessing software defection prediction performance: why using the Matthews correlation coefficient matters. In *Proceedings of the Evaluation and Assessment in Software Engineering*. 120–129.

[74] Suraj Yatish, Jirayus Jiarpakdee, Patanamon Thongtanunam, and Chakkrit Tantithamthavorn. 2019. Mining software defects: should we consider affected releases?. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 654–665.

[75] Thomas Zimmermann, Rahul Premraj, and Andreas Zeller. 2007. Predicting defects for eclipse. In *Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007)*. IEEE, 9–9.

[76] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. 2000. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation* 8, 2 (June 2000), 173–195. https://doi.org/10.1162/106365600568202

[77] Eckart Zitzler and Simon Künzli. 2004. Indicator-Based Selection in Multiobjective Search. In *International Conference on Parallel Problem Solving from Nature*. 832–842. https://doi.org/10.1007/978-3-540-30217-9_84