Why Train-and-Select When You Can Use Them All? Ensemble Model for Fault Localisation

Jeongju Sohn
School of Computing, KAIST
Daejon, Republic of Korea
kasio555@kaist.ac.kr

Shin Yoo
School of Computing, KAIST
Daejon, Republic of Korea
shin.yoo@kaist.ac.kr

ABSTRACT

Learn-to-rank techniques have been successfully applied to fault localisation to produce ranking models that place faulty program elements at or near the top. Genetic Programming has been successfully used as a learning mechanism to produce highly effective ranking models for fault localisation. However, the inherent stochastic nature of GP forces its users to learn multiple ranking models and choose the best performing one for the actual use. This train-and-select approach means that the absolute majority of the computational resources that go into the evolution of ranking models are eventually wasted. We introduce Ensemble Model for Fault Localisation (EMF), which is a learn-to-rank fault localisation technique that utilises all trained models to improve the accuracy of localisation even further. EMF ranks program elements using a lightweight, voting-based ensemble of ranking models. We evaluate EMF using 389 real-world faults in Defects4J benchmark. EMF can place 30.1% more faults at the top when compared to the best performing individual model from the train-and-select approach. We also apply Genetic Algorithm (GA) to construct the best performing ensemble. Compared to naively using all ranking models, GA generated ensembles can localise further 9.2% more faults at the top on average.

CCS CONCEPTS

• **Software and its engineering → Search-based software engineering**;

KEYWORDS

Fault Localisation, SBSE, Fitness Evaluation

ACM Reference Format:
Jeongju Sohn and Shin Yoo. 2019. Why Train-and-Select When You Can Use Them All? Ensemble Model for Fault Localisation. In *Genetic and Evolutionary Computation Conference (GECCO '19), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3321707.3321873

1 INTRODUCTION

The growing complexity of software systems, coupled with increasingly shorter release cycles, means that developers have to debug more complicated faults in shorter time. Fault localisation has been widely studied to reduce the burden of debugging: it is a technique that aims to automatically identify the program elements that are responsible for the observed failure [37]. A common approach is to rank program elements according to their relative *suspiciousness*, i.e., the relative likelihood of each program element being responsible for the failure. A wide range of techniques have been studied. Spectrum Based Fault Localisation (SBFL) uses a risk evaluation formula to convert program spectrum (the combination of test results and test coverage) to relative suspiciousness of each program element [7, 18, 25, 38]. Mutation Based Fault Localisation (MBFL), on the other hand, utilises mutation analysis to either measure the similarity between mutants and the observed failure [26], or to identify potential partial fixes [16, 24].

Recently, a growing body of work has successfully reformulated fault localisation as a learning problem: instead of manually designing a specific technique, the new approach is to try to learn the best ranking model that can place the faulty program element as near as the top, using provided failure data. Various forms of learning have been tried, including evolving individual risk evaluation formulæ [41, 42], learning the best linear combination of multiple formulæ [39], and learning ranking models that use multiple data sources such as program spectrum, invariant information, and static code features [1, 31].

Genetic Programming has been successfully applied both to evolve individual risk evaluation formulæ [41, 42][1] and to learn larger ranking models that combine multiple risk evaluation formulæ and other features about source code and change history [31]. Due to the inherent stochastic nature of GP, users of these techniques are typically expected to evolve (i.e., learn) multiple formulæ or models and choose the best performing one. We hereafter call this approach train-and-select. As with many other applications of evolutionary computation in software engineering, the train-and-select approach for fault localisation results in a large amount of wasted fitness evaluation, as the models not selected by the user are simply discarded.

The motivating question for our work is whether we can utilise these discarded models, as well as the computational resources that go into their fitness evaluation, to improve the current train-and-select approach. We introduce EMF, a voting-based ensemble technique that uses multiple ranking models to produce a single final ranking. EMF takes individual ranking models, and aggregates

[1]The human competitiveness of GP as a way of designing risk evaluation formulæ has been acknowledged by the ACM SIGEVO HUMIES Award Silver Medal in 2017.

their results using various voting schemes. The use of stochastic learn-to-rank models with the train-and-select approach results in diversity among the ranking models as a natural by-product. We wish to exploit the diversity among the otherwise discarded models. While the wasted fitness evaluation observed in GP motivates our work, EMF is not specific to GP and can take individual ranking models generated by any other learn-to-rank techniques into its voter ensemble.

Our empirical evaluation compares the performance of EMF to the results obtained by individual ranking models used by EMF. For participating individual ranking models, we implement and evaluate the state-of-the-art GP based fault localisation technique, FLUCCS, as well as models generated by Gaussian Process Models (GPM), and Support Vector Machine (SVM). We evaluate multiple voting schemes to investigate the most effective way of aggregating results of participating ranking models. Finally, we investigate whether we can generate the best ensemble, i.e., whether we can improve EMF even further by choosing the individual models that take parts in the ensemble using Genetic Algorithm (GA). All empirical evaluations are performed using the 389 real-world faults from Defects4J repository [19]. The results show that, when compared to the best individual ranking model in the whole pool, EMF can rank 30.1% more faults at the top than the ranking model.

The technical contributions of this paper are as follows:

- We introduce EMF, a new fault localisation technique that uses voting based ensemble to utilise multiple ranking models. This allows us to utilise the ranking models that would be otherwise discarded in the train-and-select approach, which is commonly adopted by both GP and other stochastic learn-to-rank techniques.
- We present an empirical study of EMF using the 389 real-world faults in Defects4J. EMF can place at most 30.1% more faults at the top, when compared to the chosen ranking model from the train-and-select approach.
- We further show that Genetic Algorithm (GA) can successfully construct the best ensemble out of the available individual ranking models. The constructed ensemble can rank 9.2% more faults at the top when compared to the default EMF model that uses all given individual models.

The rest of paper is organised as follows: Section 2 explains the overall architecture of EMF and three voting schemes of EMF. Section 3 describes the ensemble construction method using two algorithms. Section 4 introduces our research questions. Section 5 presents the set-ups for empirical evaluation, the results of which are discussed in Section 6. Potential threats to validity are considered in Section 7. Section 8 presents the related work and Section 9 concludes the paper.

2 EMF: ENSEMBLE MODEL FOR FAULT LOCALISATION

This section explains how EMF aggregates the results of individual ranking models using a voting based ensemble.

2.1 Overall architecture of EMF

EMF takes a set of individual ranking models for fault localisation as input: EMF itself consists of ranking and voting phase. The ranking phase is simply when all participating ranking models compute rankings for the program elements of the System Under Test (SUT).

In the voting phase, each ranking model casts votes for program elements based on its own ranking. EMF uses plural voting: all participating ranking models cast votes to all program elements ranked from 1 to k-th places. The reason for the plural voting scheme is as follows. The participating models are generated by learn-to-rank techniques that seek to minimise the average loss, i.e., to rank faulty program elements in the training datasets as near the top as possible. The optimisation for the average, coupled with stochastic nature of some of the learn-to-rank techniques, means that there is no guarantee that all faults will be ranked at the top. By adopting plural voting, we wish to increase the probability of the faulty program element getting at least one vote. We call any program element that has received at least one vote by participating ranking models a *candidate*. In principle, using larger k increases the probability of having the actual faulty program elements among the candidates.

2.2 Voting Schemes

The default voting scheme for EMF is that all program elements ranked up to k-th place by individual ranking models receive one vote. We call this the basic voting scheme, denoted by V_{BASIC}. EMF that uses V_{BASIC} is called as EMF_{B}, hereafter. Under V_{BASIC}, each program element m receives the following final number of votes, b_m, from the pool of all individual ranking models, $R = \{r_1, \ldots, r_n\}$:

$$b_m = \sum_{r_i \in R} v(r_i, m),\ \text{where } v(r_i, m) = \begin{cases} 1 \text{ if } r_i \text{ voted for } m \\ 0 \text{ otherwise} \end{cases}$$

We propose two additional voting schemes to deal with one weakness of the basic scheme, V_{BASIC}, which is that the scheme is prone to producing ties in the final, aggregated ranking. Ties are in general harmful to fault localisation, as it forces the user to examine more program elements when considering the ranking of program elements. The occurrence of ties with V_{BASIC} is partly because each individual ranking model gives the equal one vote to all of its k candidates. A candidate ranked higher by an individual ranking model should receive more votes than those ranked lower.

The first alternative scheme is V_{TIE}, which considers the number of tied program elements in the ranking produced by the individual model. In V_{TIE}, individual ranking models cast fewer votes to program elements that are tied. When n program elements are tied at the same rank, we assume that all n elements are equally likely to contain the fault, and make the individual ranking model to cast $\frac{1}{n}$ vote to each of the tied elements. Similarly, in V_{RANK}, individual ranking models cast $\frac{1}{j}$ vote to a program elements ranked at the j-th place.

Let $R = \{r_1, \ldots, r_n\}$ be the pool of all used ranking models: given a program element m, $r_i(m)$ returns the rank of m according to r_i, and $tie(r_i, m)$ returns the number of program elements placed at the same rank as m by r_i (including m itself). Under V_{TIE}, the number of votes m receives, b_m is defined as:

$$V_{\text{TIE}} : b_m = \sum_{r_i \in R} \frac{v(r_i, m)}{tie(r_i, m)}$$

The second alternative scheme, V_{RANK}, considers the rank assigned by the individual model. Under V_{RANK}, the number of votes

m receives, b_m is defined as:

$$V_{\text{RANK}} : b_m = \sum_{r_i \in R} \frac{v(r_i, m)}{r_i(m)}$$

V_{TIE} and V_{RANK} can be considered as variations of weighted voting. In weighted voting, each voter has a different weight, or influence, over the final decision. V_{TIE} and V_{RANK} voting scheme represent the different weights as different amount of votes. Variations of EMF using V_{TIE} and V_{RANK} are called EMF_{T} and EMF_{R}, respectively.

3 ENSEMBLE CONSTRUCTION USING GA

While the core strategy of EMF is using multiple ranking models instead of choosing a single best one, it is also possible for EMF to have *too many* participating individuals. Suppose there are several ranking models that vote for the same non-faulty program element. If the combined votes from these models outnumber the votes the real faulty element receives, the final rank of the real faulty element will be sub-optimal. An ideal ensemble is not the one that contains the largest number of participating individual ranking models, but the one whose aggregated results place the largest number of real faulty elements at the top.

To construct better performing ensembles, we apply two widely studied algorithms: greedy algorithm and Genetic Algorithm. The goal is to formulate an ensemble that would allow EMF to locate as many faults near the top as possible. Let $E \subseteq R$ be a subset of ranking models in the pool (i.e., an ensemble), and F be a set of faults to be localised. An ensemble E is scored using the number of faults in F that E places within the top n (this is a widely used evaluation metric for fault localisation, called *acc@n*: see Section 5.2 for more details). The score function ω_n for E against F is simply:

$$\omega_n(E, F) = acc@n(E, F)$$

We use ω_n to make the greedy choice for the ensemble construction, as well as the fitness function for the GA based ensemble construction (see Section 5.3).

3.1 EMF with Greedy Algorithm

Greedy algorithm aims to approximate a global optimum by making the best-at-the-time choice at each decision point greedily. Greedy algorithm has been studied in context of Next Release Problem [2, 3], as well as test suite minimisation [32]. We use ω_n function above to guide the decisions. Starting with the ranking model that produces the highest ω_n score, the greedy algorithm iteratively extends the ensemble by examining ranking models in the descending order of their ω_n scores: a new ranking model is added to the ensemble if and only if ω_n score of the ensemble increases after addition. The algorithm is therefore deterministic against a fixed pool of ranking models and a fixed set of faults, and terminates after considering all ranking models in the pool.

3.2 EMF with Genetic Algorithm

Genetic Algorithm (GA) has been successfully applied to many software engineering problems, such as test data generation [11, 33] and test case prioritisation [8, 23]. When applied to the ensemble construction, each chromosome in GA represents a single ensemble and a population of ensembles is maintained during the search. We use ω_n as the fitness function for the GA: GA evolves ensembles to have a higher fitness score and returns the ensemble with the highest fitness value after a predefined number of generations. Unlike the greedy algorithm, which is a constructive heuristic, GA is free from premature commitment to a voter (i.e., if a combination of two or more voters is more successful than having a single voter that is better than others, it is possible for GA to identify the more successful combination, while the greedy algorithm simply commits to the single voter.)

3.3 EMF with Random Construction

Both the greedy algorithm and GA aim to select a subset of ranking models (an ensemble) from the pool of all models. A random construction method, which randomly decides whether to keep each ranking model in the ensemble, is used as a baseline.

4 RESEARCH QUESTIONS

This paper poses the following research questions.

RQ1. Effectiveness: RQ1 asks whether EMF is more effective at localising faults than the train-and-select approach that trains multiple models with the aim of minimising the average loss and subsequently chooses the best performing ranking model. RQ1 is answered by comparing EMF to the ranking models chosen from the train-and-select approach using the evaluation metrics (see Section 5.2).

RQ2. Voting Scheme: RQ2 asks how much impact the different voting schemes have on the localisation effectiveness. To answer this, we compare the results of EMF_{B}, EMF_{T}, and EMF_{R} against each other.

RQ3. Ensemble Construction: RQ3 asks whether an active ensemble construction can improve the performance of EMF. The empirical evaluation compares the results from the ensembles constructed by the greedy algorithm and the GA to the results generated by the entire pool of ranking models as well as to the results of randomly constructed ensembles. We use Vargha-Delaney A_{12} statistic [34] to inspect whether the performance difference between the ensemble construction and its baselines holds statistical significance.

Table 1: Subject software systems and their faults

Project	# Faults	Loc	# Methods	# Test cases
Commons Lang	63	9059–11490	1953–2408	1540–2295
Commons Math	105	4726–41344	1049–6668	817–4429
Joda-Time	26	12732–13270	3628–3802	3749–4041
Closure Compiler	133	30438–50523	4848–8880	2595–8443
Jfreechart	26	41075–51523	6578–8281	1586–2193
Mockito	36	2110–4385	747–1476	695–1399

5 EXPERIMENTAL SETTINGS

5.1 Subjects

The empirical evaluation uses version 1.1.0 of Defects4J [19], a real-world fault benchmark, to evaluate EMF: this version contains 395 real world Java faults collected from six open source projects.

EMF performs method granularity fault localisation on Defects4J benchmark. Among 395 faults, 389 faults are actually used in the study: the remaining six faults are excluded due to the following reasons: exclusion due to failure to compile (one fault), exclusion due to faults being out of the scope of the project code base (one fault), and exclusion of the methods that cannot be directly invoked by JVM instructions such as class initialisation methods (four faults). We also exclude methods that are not executed by any test cases from the scope of localisation.

A total of 91 ranking models have been generated as individual participating models, using FLUCCS [31] with four different learn-to-rank algorithms: Genetic Programming (GP), 30 with Gaussian Process Modelling (GPM), and Support Vector Machine (SVM). With stochastic algorithms (GP, GPM, and RF), we train 30 different ranking models. SVM, on the other hand, is deterministic and produces a single ranking model. For GP, DEAP [9], a Python evolutionary computation framework, is used. We use `GPy` [13] for GPM and `sklearn` [4] version 0.19.1 for RF. RankSVM, which depends on LibSVM [5], is used for SVM. With GP, we evolve a formula with which we rank program elements, using the average rank of faulty program elements in the training data as the fitness. With GPM, RF, and SVM, we perform the ordinal regression by first training classifiers and subsequently using the probability of each of the program elements to be faulty to rank them.

5.2 Evaluation metric

We use *acc@n* and wef^* for the analysis of EMF, following existing work [1, 31, 39]. The use of these metrics follows the guideline suggested by Parnin and Orso [1, 27], describing the actual efforts and time needed for the the target fault. In EMF, the fault is considered being localised if it is included in the final output of EMF, which is a list of candidates sorted in descending order of their obtained votes. We include the number of localised faults (nfc) as an additional evaluation metric, for EMF does not guarantee the localisation of faults.

5.2.1 *Accuracy (acc@n).*

acc@n counts the number of faults localised within top *n* elements. For *n*, we use 1, 3, and 5. If there are more than one element responsible for the fault, we use the one ranked the highest to compute *acc@n*.

5.2.2 *Wasted Effort (wef^*).*

The traditional wef metric measures the amount of effort wasted on inspecting non-faulty elements; it counts the number of non-faulty elements that developers have to examine before meeting the faulty one. Similar to *acc@n*, when there are multiple faulty elements, we only consider the one ranked the highest. Because the final result produced by EMF is a ranking of candidates only, it is possible that the actual faulty program element is not included in the result. Hence we use wef^*, which is wef measured against the ranking of the candidates.

5.2.3 *Number of Faults among Candidates (nfc).*

wef^* alone does not reflect whole localisation effectiveness, as EMF may be entirely missing a fault from its final result of candidate ranking. To complement wef^*, we measure how frequently EMF succeeds to include the actual faulty elements in its ranking, using nfc. nfc counts the number of faults that are also candidates.

5.3 Configuration

We use ω_1 for both the greedy and GA-based ensemble construction, i.e., we seek to construct an ensemble that maximises *acc*@1. For all construction algorithms, we use a binary string of length 91 to represent the ensemble. GA-based construction has been implemented using DEAP [9], a Python evolutionary computation framework, to implement GA. We use the consecutive mating[2], single point crossover with crossover rate of 0.8, and single bit-flip mutation with mutation rate of 0.2. The population size of GA is 64; when formulating the next generation of population, we choose the best 64 individuals out of the combined parent and offspring generations containing 128 individuals. The stopping criterion is after 50 generations.

5.4 Validation

To avoid overfitting and to make the most of the given training data, we use ten fold cross validation. We divide the 389 faults into ten folds as evenly as possible. Subsequently, each fold becomes the test data when evaluating EMF trained using the remaining nine folds. The entire process from the training of the individual ranking models to the final voting of EMF and the ensemble construction is conducted as a ten fold cross validation.

Due to the stochastic nature of GA and random, we repeat the ensemble construction 20 times for both algorithms. Evaluation on these algorithms will be performed on the average of the results from these 20 repetitions.

6 RESULTS AND ANALYSIS

This section presents the results of our empirical evaluation along with answers to research questions set in Section 4.

6.1 RQ1. Effectiveness

Table 2 contains both the results of the best ranking model with the smallest average wef labelled `BEST`[3], and the results of EMF_B^k with $k = 1, 3, 5$ to align the ensembles with our choice of evaluation metric, *acc@n* with $n = 1, 3, 5$. Fig. 1 presents details about the number of faults localised at each of the top ten ranks. Overall, EMF_B outperforms the best ranking model at locating faults within higher ranks regardless of *k*, placing up to 28 (19.6%) more faults at the top (EMF_B^1). On the other hand, nfc for EMF_B^1 is 308, which means that 81 faults (20.8% of the total) are missing from the results. This is because EMF only ranks the candidates, i.e., the program elements with at least one vote. If no participating ranking model votes for the root cause of a specific fault, EMF cannot include it in its ranking. However, we argue that actual faulty program elements that are not candidates will be equally harder to localise for individual ranking models, and EMF does not introduce any additional weakness.

Note that nfc increases as we increase *k*: nfc for EMF_B^1 is 308, but nfc for EMF_B^5 is 366. A larger *k* naturally results in more candidates, which in turn increases the probability of the faulty program

[2]Following a random order in the population, pairs of two consecutive individuals are selected as parents: this is effectively the `varAnd` implementation in DEAP (see https://deap.readthedocs.io/en/master/api/algo.html).
[3]Out of ten best ranking models selected, nine are GP models, and one is an SVM model.

Table 2: Evaluation results of $\mathrm{EMF}_\mathrm{B}^k$, $\mathrm{EMF}_\mathrm{T}^k$, and $\mathrm{EMF}_\mathrm{R}^k$ (k = 1, 3, and 5). Unlike $\mathrm{EMF}_\mathrm{B}^k$, for which $acc@1$ decreases as k increases, all $acc@n$ values increase along with k for $\mathrm{EMF}_\mathrm{T}^k$ and $\mathrm{EMF}_\mathrm{R}^k$. $\mathrm{EMF}_\mathrm{T}^5$ shows the best performance in $acc@1$, $acc@3$, and wef^*.

Project	BEST						V_{BASIC}						V_{TIE}						V_{RANK}					
(# Faults)	acc			wef^*			acc			wef^*		nfc	acc			wef^*		nfc	acc			wef^*		nfc
	@1	@3	@5	mean	std	k	@1	@3	@5	mean	std		@1	@3	@5	mean	std		@1	@3	@5	mean	std	
						1	40	55	55	0.75	2.46	57	40	53	55	0.61	1.41	57	40	55	55	0.75	2.46	57
Lang (63)	34	48	53	2.43	5.72	3	34	53	57	2.43	8.64	63	39	53	57	1.40	2.91	63	40	54	58	1.33	3.14	63
						5	36	56	58	2.30	7.98	63	40	56	58	1.32	3.47	63	40	53	58	1.22	2.99	63
						1	45	63	74	5.72	15.77	88	45	67	78	3.20	9.32	88	45	64	75	3.22	8.25	88
Math (105)	46	61	69	13.40	42.13	3	41	65	75	8.89	21.67	96	46	66	75	4.56	11.26	96	47	66	76	4.86	13.47	96
						5	40	65	73	10.75	23.78	99	46	67	73	5.39	13.43	99	48	66	76	6.34	16.44	99
						1	11	17	19	0.79	1.06	19	11	17	19	0.74	1.02	19	11	17	19	0.79	1.06	19
Time (26)	10	16	18	8.85	14.33	3	12	18	19	6.22	14.32	23	11	18	19	3.04	5.35	23	11	19	19	4.13	8.63	23
						5	10	17	20	7.04	16.05	24	14	17	20	3.79	7.15	24	11	19	20	4.71	9.92	24
						1	49	71	80	3.76	10.36	94	50	71	79	2.57	5.14	94	49	71	80	3.22	7.17	94
Closure (133)	28	67	76	19.66	63.45	3	52	73	87	6.91	19.56	110	53	74	86	4.62	10.30	110	50	75	88	4.71	11.67	110
						5	54	78	88	13.53	32.34	118	57	80	87	8.64	19.76	118	49	76	90	9.99	24.46	118
						1	18	20	21	0.24	0.68	21	18	20	21	0.24	0.68	21	18	20	21	0.24	0.68	21
Chart (26)	17	22	24	1.27	3.37	3	16	22	22	1.00	1.98	25	17	22	22	0.84	1.62	25	18	22	22	1.12	2.96	25
						5	15	23	25	3.35	13.36	26	16	22	25	1.08	2.23	26	18	22	24	1.58	4.73	26
						1	8	18	21	5.83	10.99	29	9	20	21	3.62	5.88	29	8	18	21	5.83	10.99	29
Mockito (36)	8	18	23	26.64	122.69	3	12	23	26	10.50	30.74	34	14	23	26	5.38	11.47	34	9	22	26	6.09	12.18	34
						5	11	20	24	16.00	58.41	36	13	21	23	7.06	14.77	36	11	22	25	7.11	12.12	36
						1	171	244	270	3.56	10.78	308	173	248	273	2.25	6.01	308	171	245	271	2.70	6.95	308
Total (389)	143	232	263	13.87	57.69	3	167	254	286	6.61	19.27	351	180	256	285	3.80	9.28	351	175	258	289	4.06	10.79	351
						5	166	259	288	10.08	29.42	366	186	263	286	5.61	14.43	366	177	258	293	6.41	17.34	366

elements being one of the candidates. However, since the V_{BASIC} voting scheme assigns one vote to all candidates, regardless of their original ranking generated by individual ranking models, the larger nfc does not always lead to more faults ranked at the top. With more candidates, the actual faulty program element may get tied with others, or even ranked lower than common mistakes. The results support this suspicion: the total $acc@1$ is 171 when $k = 1$, but it decreases to 166 when $k = 5$. We try to fix this using V_{TIE} and V_{RANK}, the results of which are discussed in Section 6.2.

Answer to **RQ1**: EMF_B outperforms the best individual ranking model, placing 28 (19.6%) more faults at the top. By using a larger k value, EMF_B overcomes its inherent weakness in nfc while still excels in placing more faults within higher ranks, compared to the best ranking model.

6.2 RQ2. Voting Scheme

Table 2 presents the results of EMF with $\mathrm{EMF}_\mathrm{T}^k$ and $\mathrm{EMF}_\mathrm{R}^k$ with $k = 1, 3, 5$. While there are a few configurations that perform worse than their counterpart EMF_B (e.g., $\mathrm{EMF}_\mathrm{T}^k$ with $k = 3, 5$ produces lower $acc@5$, and $\mathrm{EMF}_\mathrm{R}^5$ produces lower $acc@3$), both EMF_T and EMF_R improve performance compared to EMF_B in most cases; e.g., $\mathrm{EMF}_\mathrm{T}^5$ localises 20 more faults (12.0%) at the top compared to $\mathrm{EMF}_\mathrm{B}^5$. The mean wef^* also decreases for both EMF_T and EMF_R with all k values studied.

Another improvement that has been shown in both EMF_T and EMF_R is that $acc@1$ no longer decreases when using a larger k. On the contrary, EMF_T and EMF_R rank more faults at the top from using a larger k: $\mathrm{EMF}_\mathrm{T}^5$ places 13 (7.5%) more faults at the top than $\mathrm{EMF}_\mathrm{T}^1$, and $\mathrm{EMF}_\mathrm{R}^5$ places 6 (3.5%) more faults at the top. Compared to BEST, the amount of performance improvement EMF_T and EMF_R provide is even greater, as they place 43 (30.1%) and 34 (23.8%) more faults at the top with $k = 5$, respectively.

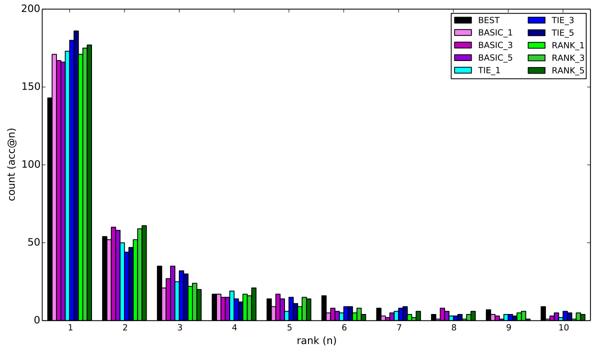

Figure 1: `BASIC_k`, `TIE_k`, `RANK_k`, and `BEST` represent $\mathrm{EMF}_\mathrm{B}^k$, $\mathrm{EMF}_\mathrm{T}^k$, $\mathrm{EMF}_\mathrm{R}^k$, and the ranking model selected from the train-and-select approach using wef as a loss function, respectively. Each bar in the histogram indicates the number of faults placed at a specific rank (x axis) from using a specific variation of EMF. The order of EMF's variations is `BEST`, `BASIC_k` with k = 1, 3, 5, `TIE_k` with k = 1, 3, 5, and `RANK_k` with k = 1, 3, 5. Generally, EMF outperforms `BEST`, placing more faults at the top. `TIE_k` and `RANK_k` improve the effectiveness of EMF even further.

While both EMF_T and EMF_R produce significant improvement over EMF_B, we note that the difference between these two models are of a smaller scale. EMF_T tends to place more faults at the top when compared to EMF_R; for example, $\mathrm{EMF}_\mathrm{T}^5$ puts nine more faults at the top than $\mathrm{EMF}_\mathrm{R}^5$. Even when $\mathrm{EMF}_\mathrm{R}^3$ and $\mathrm{EMF}_\mathrm{R}^5$ produce higher $acc@5$, their mean wef^* values are larger than those of the corresponding EMF_T configurations. Fig. 1 presents an overview of the impact of V_{TIE} and V_{RANK}. Compared to EMF_B, both EMF_T and EMF_R put more faults at the top, which is likely due to the fact that these voting schemes are less prone to generate ties.

Answer to **RQ2**: Both EMF_T and EMF_R improve the performance of EMF by localising at most 20 (12.0%) more faults at the top. Compared to the best individual ranking model from the train-and-select process, EMF^5_T increases *acc*@1 by 43 (30.1%), highlighting the benefit of EMF over the train-and-select approach. Between EMF_T and EMF_R, EMF_T ranks more faults at, or near, the top, generally resulting in larger *acc*@1 and smaller wef^* than EMF_R.

Table 3: Evaluation results of EMF^k_T and EMF^k_R with the ensembles constructed by the greedy algorithm. Compared to Table 2, the greedy algorithm fails to formulate better performing ensembles for EMF^1_T, EMF^1_R, and EMF^3_R. For EMF^3_T, it finds a relatively effective ensemble that increases *acc*@1, *acc*@3, and *acc*@5.

Project (# Faults)		V_TIE						V_RANK					
		acc			wef^*		nfc	*acc*			wef^*		nfc
	k	@1	@3	@5	mean	std		@1	@3	@5	mean	std	
Lang (63)	1	41	44	44	0.20	0.81	45	41	44	45	0.16	0.56	45
	3	39	55	57	1.21	2.63	63	41	50	54	0.70	1.47	57
Math (105)	1	39	48	51	0.81	1.81	54	36	46	48	0.64	1.38	50
	3	52	70	78	2.61	5.62	92	40	62	67	2.83	7.44	76
Time (26)	1	13	15	15	0.13	0.34	15	11	15	15	0.27	0.44	15
	3	15	18	19	1.38	3.21	21	13	19	19	0.32	0.46	19
Closure(133)	1	48	52	53	0.30	1.18	54	47	50	51	0.25	1.05	52
	3	55	76	90	2.83	7.38	104	48	63	70	2.00	4.25	81
Chart (26)	1	8	12	14	0.79	1.08	14	8	12	14	0.79	1.08	14
	3	16	22	24	1.24	3.37	25	10	16	19	1.24	1.60	21
Mockito (36)	1	7	9	13	2.36	3.87	14	7	9	12	2.43	3.90	14
	3	11	18	24	3.97	5.29	33	7	12	18	3.52	4.47	23
Total (389)	1	156	180	190	0.63	1.71	196	150	176	185	0.58	1.56	190
	3	188	259	292	2.41	5.69	338	159	222	247	1.99	4.85	277

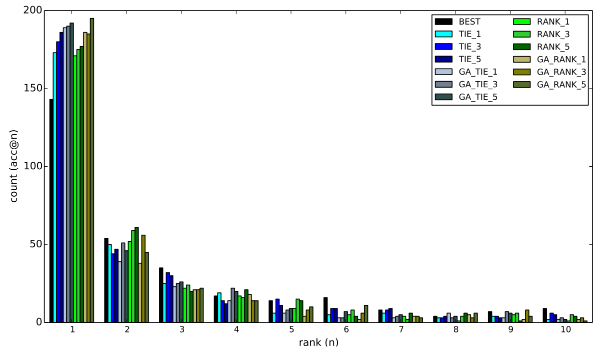

Figure 2: Each bar in the histogram indicates the number of faults placed at a specific rank (x axis) from a specific variation of EMF. Only the results of ensemble construction via GA are presented here. The order of EMF's variations in the histogram is `BEST`, `TIE_k` with $k = 1, 3, 5$, `TIE_k`'s with ensemble construction (`GA_TIE_k`), `RANK_k` with $k = 1, 3, 5$, and `RANK_k` with ensemble construction (`GA_RANK_k`). Overall, GA formulates more effective ensembles, pushing up faults placed at lower ranks to higher ranks.

6.3 RQ3. Ensemble Construction

Table 3 and Table 4 describe the results of ensemble construction using the greedy algorithm and the genetic algorithm respectively. Due to the space restriction, the paper only contains the results with $k = 1$ and 3: the full results can be found at https://figshare.com/articles/EMF_pdf/7665467.

Table 4: Evaluation results of EMF^k_T and EMF^k_R with the ensembles constructed by GA. Compare to Table 2, *acc*@1/3/5 and *acc*@1/3 increase for EMF^k_T and EMF^k_R with all k values studied, respectively.

Project (# Faults)		V_TIE					
		acc			wef^*		nfc
	k	@1	@3	@5	mean	std	
Lang (63)	1	42.95	54.25	54.7	0.54	1.57	56.95
	3	41.7	54.95	57.3	1.22	2.91	62.95
Math (105)	1	49.85	70.8	77.0	2.85	9.55	85.9
	3	51.6	71.65	78.55	3.77	10.40	94.0
Time (26)	1	12.2	18.1	18.9	0.60	0.92	18.9
	3	13.6	18.9	19.0	1.58	3.52	21.3
Closure (133)	1	56.6	71.75	79.7	2.20	5.39	91.35
	3	55.5	77.95	89.8	3.85	8.93	108.95
Chart (26)	1	16.75	20.0	20.6	0.38	1.04	21.0
	3	15.65	22.0	23.6	0.91	1.93	25.0
Mockito (36)	1	10.6	16.7	20.25	2.85	3.78	26.8
	3	11.65	19.8	25.5	3.86	6.68	32.8
Total (389)	1	188.95	251.6	271.15	1.94	6.03	300.9
	3	189.7	265.25	293.75	3.06	7.94	345.0

Project (# Faults)		V_RANK					
		acc			wef^*		nfc
	k	@1	@3	@5	mean	std	
Lang (63)	1	42.85	54.7	54.95	0.63	2.17	56.9
	3	41.95	54.85	57.3	1.33	3.50	63.0
Math (105)	1	48.9	70.25	75.5	3.05	9.22	85.6
	3	49.75	72.35	78.3	3.64	9.98	93.85
Time (26)	1	12.45	18.05	18.6	0.54	0.90	18.7
	3	13.2	18.95	19.05	1.96	4.95	21.3
Closure (133)	1	54.7	70.0	78.05	2.89	6.89	91.45
	3	56.6	75.75	85.4	4.34	10.74	108.85
Chart (26)	1	15.75	19.75	20.85	0.41	0.88	21.0
	3	16.55	22.0	23.65	1.22	3.31	24.95
Mockito (36)	1	10.8	16.0	19.55	4.25	6.57	27.3
	3	11.5	17.25	22.15	5.12	7.94	32.6
Total (389)	1	185.45	248.75	267.5	2.37	6.64	300.95
	3	189.55	261.15	285.85	3.37	8.75	344.55

From the results, we conclude that the greedy construction produces mixed results. While the greedy construction improves *acc*@1, *acc*@3, and *acc*@5 of EMF^3_T respectively by eight (4.4%), three (1.2%), and seven (2.5%) when compared to using all ranking models, *acc*@*n* results deteriorate in all other cases. We do note that wef^* decreases, but suspect that this is because the ensembles constructed by greedy are failing to include harder-to-localise faults in their results, as evidenced by lower nfc values. Since the greedy algorithm is deterministic, it produces a single ensemble for each of the six configurations (V_TIE and V_RANK, each with $k = 1, 3, 5$), resulting in six ensembles. The sizes of these ensembles range from 1 to 23, the mean being 12.5.

Table 4 presents the results of GA based ensemble construction: due to stochasticity, all values in Table 4 are the average of 20 repetitions. Compared to the results without ensemble construction in Table 2, *acc*@1 and *acc*@3 of both EMF_T and EMF_R increase. While the improvements in *acc*@1 and *acc*@3 have been observed at both k values (1 and 3), the benefit of applying the ensemble construction is more noticeable when $k = 1$, especially in the case of EMF_T: on average, 15.95 (9.2%) more faults are placed at the top (*acc*@1) by EMF^1_T when using the ensemble constructed by GA. The ensemble improves *acc*@5 of EMF_T^3 by 8.75 (3.1%). However, overall, the differences in *acc*@5 are relatively small, ranking between 1.85 and 3.5 faults except in the case of EMF_T^3. In general, the advantage of using the GA based ensemble construction becomes less noticeable as n of *acc*@n increases. We repeat GA based ensemble construction 20 times for each of the six configurations (V_TIE and V_RANK, each with $k = 1, 3, 5$), resulting 120 ensembles. The sizes of these ensembles range from 25 to 44, the mean being 34.4.

We note that the increase of *acc*@1 for EMF_T^1 comes at the cost of the decrease in nfc by 8.1. Since constructed ensembles are more selective (i.e. fewer participating ranking models), they tend to produce fewer candidates, which results in lower nfc. However, our expectation with constructed ensembles is to filter out *misinformed voters* that may promote non-faulty program elements higher than the actually faulty one, rather than to localise more faults. The increase in *acc*@1 suggests that our motivation for ensemble construction (see Section 3) is justified.

Comparison with the random baseline, presented in Table 6, further shows that GA is capable of constructing better ensembles. The ensembles constructed by GA outperforms random ensembles with respect to both *acc*@n ($n = 1, 3, 5$) and nfc: EMF_R^3 ranks up to 16.65 (9.6%) more faults at the top when using ensembles constructed by GA.

When compared to results obtained using the entire pool of ranking models in Table 2, the performance of random ensembles is comparable to the performance of all ranking modes in both EMF_T and EMF_R: the difference in *acc*@n is at most 6.95 (*acc*@5 in EMF_R^1), and is equal to or smaller than three in two thirds of the cases. We repeat random ensemble construction 20 times for each of the six configurations (V_TIE and V_RANK, each with $k = 1, 3, 5$), resulting 120 ensembles. The sizes of these ensembles range from 36 to 52, the mean being 46.9.

Table 5 presents the effect size of *acc*@1 measured in A_{12} statistic. Here, an effect size larger than 0.5 means the ensemble construction shows better performance than its baselines, which are either using all ranking models (*all*) or random ensemble construction (*random*). Since the *all* configuration and the greedy ensemble construction are deterministic, we repeat their *acc*@1 20 times to compute the A_{12} statistic. The results show that A_{12} effect sizes are all significantly higher than 0.5 for GA based ensembles, suggesting that the improvements are statistically significant.

Answer to **RQ3**: GA can successfully construct ensembles that improve EMF: GA based ensembles can place, on average, 15.95 (9.2%) and 16.65 (9.6%) more faults at the top than the entire pool of ranking models and randomly constructed ensembles, respectively. The effect sizes measured by A_{12} statistic show that the the improvements are statistically significant.

7 THREATS TO VALIDITY

Threats to internal validity concern whether our implementation and experiments are done correctly. The fault data we used are from Defects4J, which have been widely investigated in many studies [1, 30, 43]. The learn-to-rank algorithms are all based on existing open source frameworks [5, 9, 13, 17]. For statistical analysis, we use R version 3.4.4 with package `effsize` version 0.7.1.

Table 5: Comparing performance of EMF with the ensemble construction using either the greedy algorithm or the GA to its baselines, EMF without ensemble construction (`all`) and with random ensemble construction (`random`), by measuring effect sizes of *acc*@1 in A_{12} statistic. The ensemble construction with GA outperforms `all` and `random` in all cases.

Voting scheme	k	Greedy Algorithm vs. all	Greedy Algorithm vs. random	GA vs. all	GA vs. random
V_TIE	1	0.0000	0.0000	1.0000	1.0000
	3	1.0000	1.0000	1.0000	0.9962
V_RANK	1	0.0000	0.0000	1.0000	1.0000
	3	0.0000	0.0000	1.0000	1.0000

Table 6: EMF_T^k and EMF_R^k with the random construction: *acc*@n and nfc show slight decrease when compared to Table 2, which contains the results of using all ranking models.

Project (# Faults)		V_TIE *acc* @1	*acc* @3	*acc* @5	wef^* mean	wef^* std	nfc
Lang (63)	1	40.1	53.0	54.45	0.58	1.43	56.25
	3	37.65	53.05	56.95	1.48	4.04	62.7
Math (105)	1	45.65	65.6	75.95	2.99	10.44	84.7
	3	44.2	64.8	75.1	4.40	11.85	93.5
Time (26)	1	10.95	16.95	18.5	0.79	1.20	18.6
	3	11.75	18.15	19.0	2.33	5.31	21.6
Closure (133)	1	49.9	70.15	77.85	1.73	3.71	87.45
	3	51.45	74.05	86.0	3.35	7.48	105.7
Chart (26)	1	18.0	20.0	20.25	0.35	1.23	20.95
	3	16.6	21.8	22.75	0.83	1.60	24.85
Mockito (36)	1	7.85	19.85	21.95	2.51	3.95	26.5
	3	13.65	22.9	26.1	3.75	7.43	32.4
Total (389)	1	172.45	245.55	268.95	1.80	6.01	294.45
	3	175.3	254.75	285.9	3.14	8.18	340.75

Project (# Faults)		V_RANK *acc* @1	*acc* @3	*acc* @5	wef^* mean	wef^* std	nfc
Lang (63)	1	39.95	53.65	54.45	0.62	1.75	56.25
	3	39.9	53.4	57.05	1.36	3.35	62.7
Math (105)	1	44.95	63.1	72.85	3.73	11.35	84.7
	3	44.9	64.95	74.55	4.67	13.13	93.5
Time (26)	1	10.85	17.15	18.45	0.77	1.14	18.6
	3	10.8	18.95	19.0	2.50	5.55	21.6
Closure (133)	1	49.5	70.1	77.5	2.04	4.85	87.45
	3	50.05	74.9	87.05	3.48	8.53	105.7
Chart (26)	1	17.95	19.95	20.4	0.36	1.27	20.95
	3	17.9	21.95	22.6	1.08	2.89	24.85
Mockito (36)	1	7.75	18.5	20.4	3.58	6.15	26.5
	3	9.35	22.05	25.35	4.30	7.66	32.4
Total (389)	1	170.95	242.45	264.05	2.21	6.86	294.45
	3	172.9	256.2	285.6	3.31	9.02	340.75

One inherent limitation of EMF is that it may completely fail to include some actual faulty program elements in its results. This is because EMF only includes candidates ranked sufficiently high by individual ranking models in its result. The nfc results in Section 6.3 show that this does happen. However, overall, we argue that the improvements in the number of correctly localised faults compensate for this loss.

Threats to external validity in EMF include any potential bias in the selection of subjects, faults, and algorithms used in EMF. While 389 faults from Defects4J are from real-world, these faults are all written in Java, restricting our claim only to Java faults. EMF uses four different learning algorithms, GP, GPM, SVM, and RF. While these algorithms have been investigated in the context of learn-to-rank techniques [12, 14, 21, 41], other learn-to-rank algorithms may have different impact on the performance of ensembles. The results of GA based ensemble construction depend on our choice of GA configuration and parameters. While we cannot guarantee that our choice of GA parameters is the optimal one, our main purpose here is to show that the performance of EMF can be further improved by ensemble construction via GA, and not to tune the GA to its optimal performance.x

Threat to construct validity has to do with whether our evaluation are related to the claim that we intend to validate. We use absolute metrics, such as $acc@n$ and wef, to measure the performance of EMF. These metrics are count-based and thought to be closely correlated to the actual localisation effort required from human developers [27].

8 RELATED WORK

The core idea behind EMF is closely related to fault localisation as a learn-to-rank problem, and to various attempts to maximise the utility of fitness evaluations in SBSE.

8.1 Fault Localisation Techniques

Spectrum Based Fault Localisation (SBFL) has been widely studied in fault localisation [37]. A variety of SBFL formulæ, including Tarantula, have been developed [7, 25, 36]. In addition to manually designed SBFL formulæ, Yoo used GP to evolve SBFL formulæ automatically [41]; the technique was repeated 30 times to accommodate GP's inherent stochastic nature. Recently, it has been proven theoretically that there is no greatest SBFL formula that can outperform all the others [42]. Reflecting this theoretical insights, many existing fault localisation works use multiple SBFL formulæ. Xuan and Monperrus suggested a new fault localisation technique called MULTRIC, which learns how to combine 25 well-known SBFL formulæ using RankBoost [39]. Due to the stochastic nature of RankBoost, MULTRIC has been repeated 30 times. Sohn and Yoo introduced FLUCCS, a GP-based learn-to-rank fault localisation technique that uses multiple SBFL formulæ as well as code and change metrics measured from the code under debugging [31]; Similar to [39, 41, 42], GP-based FLUCCS was run 30 times to adjust for its randomness. EMF is highly relevant to existing work that adopts the train-and-select approach, such as MULTRIC or FLUCCS.

While the current evaluation of EMF focuses on input features used by FLUCCS (i.e., program spectrum as well as code and change metrics), other techniques such as Information Retrieval (IR) [22, 37, 40], mutation analysis [16, 26], invariant mining [1], and program slicing [35] have been applied to fault localisation. Our future work will consider a wider range of participating ranking models for EMF.

8.2 Efficient Fitness Evaluations

In many applications of evolutionary computation, the major cost is the computational cost of the fitness evaluations that are spent to generate the single final solution. Consequently, there have been many attempts to reduce this cost. Fitness inheritance tries to assign, to an offspring, a fitness value that is not computed but inherited from its parents [6, 28, 29]. Cluster-based fitness evaluation first clusters individuals into several clusters, and computes fitness for only representatives of each cluster [20]. Compared to these, the aim of EMF is not to reduce the number of fitness evaluations, but to exploit already spent fitness evaluations to a better use.

In search based test data generation, *collateral coverage* refers to any structural coverage that has been achieved as a collateral while trying to cover a specific target. Harman et al., prioritised the order of source code branches with the hope of maximising the collateral coverage [15]. EvoSuite, a widely studied test data generation tool for Java, similarly seeks to actively exploit collateral coverage, so that it can generate smaller test suites with fewer fitness evaluations [10, 11]. The difference with EMF is that fault localisation does not have multiple targets (such as branches) that can be achieved as a collateral. Rather, EMF exploits the diversity that is the by-product of the stochasticity of learn-to-rank techniques.

9 CONCLUSION

We present EMF, a new fault localisation technique that utilises ranking models that would have been discarded if the user only kept the single best performing model. Instead, EMF uses a voting based ensemble model to utilise all models that are evolved or learnt. Through multiple voting schemes, EMF exploits the diversity in multiple models, and improves the accuracy of fault localisation at no cost. Furthermore, EMF can boost the localisation accuracy even further by constructing better performing ensembles using GA. The empirical evaluation of EMF, using 389 real-world faults from Defects4J, shows that it can significantly outperform the best model chosen from the train-and-select approach. Future work will consider a wider range of fault localisation techniques to be used with EMF.

ACKNOWLEDGEMENT

This work was supported by the Next-Generation Information Computing Development Program through the National Research Foundation of Korea funded by the Korean government (MSIT) (2017M3C4A7068179), as well as the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (Grant No. NRF- 2016R1C1B1011042).

REFERENCES

[1] Tien-Duy B. Le, David Lo, Claire Le Goues, and Lars Grunske. 2016. A Learning-to-rank Based Fault Localization Approach Using Likely Invariants. In *Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA 2016)*. ACM, New York, NY, USA, 177–188.

[2] A.J. Bagnall, V.J. Rayward-Smith, and I.M. Whittley. 2001. The next release problem. *Information and Software Technology* 43, 14 (Dec. 2001), 883–890.

[3] P. Baker, M. Harman, K. Steinhofel, and A. Skaliotis. 2006. Search Based Approaches to Component Selection and Prioritization for the Next Release Problem. In *2006 22nd IEEE International Conference on Software Maintenance*. 176–185. https://doi.org/10.1109/ICSM.2006.56

[4] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort,

Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. 2013. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 108–122.

[5] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A Library for Support Vector Machines. *ACM Trans. Intell. Syst. Technol.* 2, 3, Article 27 (May 2011), 27 pages.

[6] Jian-Hung Chen, David E Goldberg, Shinn-Ying Ho, and Kumara Sastry. 2002. Fitness inheritance in multi-objective optimization. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann Publishers Inc., 319–326.

[7] Valentin Dallmeier, Christian Lindig, and Andreas Zeller. 2005. Lightweight bug localization with AMPLE. In *Proceedings of the sixth international symposium on Automated analysis-driven debugging (AADEBUG'05)*. ACM, New York, NY, USA, 99–104.

[8] Michael G. Epitropakis, Shin Yoo, Mark Harman, and Edmund K. Burke. 2015. Empirical Evaluation of Pareto Efficient Multi-objective Regression Test Case Prioritisation. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis (ISSTA 2015)*. ACM, New York, NY, USA, 234–245.

[9] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research* 13 (July 2012), 2171–2175.

[10] Gordon Fraser and Andrea Arcuri. 2011. EvoSuite: Automatic Test Suite Generation for Object-oriented Software. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE '11)*. ACM, New York, NY, USA, 416–419. https://doi.org/10.1145/2025113.2025179

[11] Gordon Fraser and Andrea Arcuri. 2013. Whole Test Suite Generation. *IEEE Trans. Softw. Eng.* 39, 2 (Feb. 2013), 276–291.

[12] Pierre Geurts and Gilles Louppe. 2010. Learning to Rank with Extremely Randomized Trees. In *Proceedings of the 2010 International Conference on Yahoo! Learning to Rank Challenge - Volume 14 (YLRC'10)*. JMLR.org, 49–61.

[13] GPy. since 2012. GPy: A Gaussian process framework in python. http://github.com/SheffieldML/GPy. (since 2012).

[14] John Guiver and Edward Snelson. 2008. Learning to Rank with SoftRank and Gaussian Processes. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '08)*. ACM, New York, NY, USA, 259–266.

[15] M. Harman, Sung Gon Kim, K. Lakhotia, P. McMinn, and Shin Yoo. 2010. Optimizing for the Number of Tests Generated in Search Based Test Data Generation with an Application to the Oracle Cost Problem. In *Proceedings of the 3rd International Workshop on Search-Based Software Testing (SBST 2010)*. 182 –191.

[16] Shin Hong, Byeongcheol Lee, Taehoon Kwak, Yiru Jeon, Bongsuk Ko, Yunho Kim, and Moonzoo Kim. 2015. Mutation-Based Fault Localization for Real-World Multilingual Programs (T). In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*. 464–475.

[17] Eric Jones, Travis Oliphant, Pearu Peterson, et al. 2001–. SciPy: Open source scientific tools for Python. (2001–). http://www.scipy.org/ [Online; accessed <today>].

[18] James A. Jones, Mary Jean Harrold, and John T. Stasko. 2001. Visualization for Fault Localization. In *Proceedings of ICSE Workshop on Software Visualization*. 71–75.

[19] René Just, Darioush Jalali, and Michael D. Ernst. 2014. Defects4J: A Database of Existing Faults to Enable Controlled Testing Studies for Java Programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis (ISSTA 2014)*. ACM, New York, NY, USA, 437–440.

[20] Hee-Su Kim and Sung-Bae Cho. 2001. An efficient genetic algorithm with less fitness evaluation by clustering. In *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*, Vol. 2. 887–894 vol. 2.

[21] Tzu-Ming Kuo, Ching-Pei Lee, and Chih-Jen Lin. 2014. Large-scale Kernel RankSVM. In *Proceedings of the 2014 SIAM International Conference on Data Mining, Philadelphia, Pennsylvania, USA, April 24-26, 2014*. 812–820.

[22] Tien-Duy B. Le, Richard J. Oentaryo, and David Lo. 2015. Information Retrieval and Spectrum Based Bug Localization: Better Together. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015)*. ACM, New York, NY, USA, 579–590.

[23] Zheng Li, Mark Harman, and Robert M. Hierons. 2007. Search Algorithms for Regression Test Case Prioritization. *IEEE Transactions on Software Engineering* 33, 4 (2007), 225–237.

[24] Seokhyeon Moon, Yunho Kim, Moonzoo Kim, and Shin Yoo. 2014. Ask the Mutants: Mutating Faulty Programs for Fault Localization. In *Proceedings of the 7th International Conference on Software Testing, Verification and Validation (ICST 2014)*. 153–162.

[25] Lee Naish, Hua Jie Lee, and Kotagiri Ramamohanarao. 2011. A model for spectra-based software diagnosis. *ACM Transactions on Software Engineering Methodology* 20, 3, Article 11 (August 2011), 32 pages.

[26] Mike Papadakis and Yves Le Traon. 2015. Metallaxis-FL: mutation-based fault localization. *Journal of Software Testing, Verification and Reliability* 25, 5-7 (2015), 605–628.

[27] Chris Parnin and Alessandro Orso. 2011. Are automated debugging techniques actually helping programmers?. In *Proceedings of the 2011 International Symposium on Software Testing and Analysis (ISSTA 2011)*. ACM, New York, NY, USA, 199–209.

[28] Martin Pelikan and Kumara Sastry. 2004. Fitness inheritance in the Bayesian optimization algorithm. In *Genetic and Evolutionary Computation Conference*. Springer, 48–59.

[29] Robert E Smith, Bruce A Dike, and SA Stegmann. 1995. Fitness inheritance in genetic algorithms. In *Proceedings of the 1995 ACM symposium on Applied computing*. ACM, 345–350.

[30] Victor Sobreira, Thomas Durieux, Fernanda Madeiral Delfim, Martin Monperrus, and Marcelo de Almeida Maia. 2018. Dissection of a Bug Dataset: Anatomy of 395 Patches from Defects4J. *CoRR* abs/1801.06393 (2018). arXiv:1801.06393 http://arxiv.org/abs/1801.06393

[31] Jeongju Sohn and Shin Yoo. 2017. FLUCCS: Using Code and Change Metrics to Improve Fault Localisation. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA 2017)*. 273–283.

[32] Sriraman Tallam and Neelam Gupta. 2006. A concept analysis inspired greedy algorithm for test suite minimization. *SIGSOFT Software Engineering Notes* 31, 1 (2006), 35–42.

[33] Paolo Tonella. 2004. Evolutionary Testing of Classes. In *Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '04)*. ACM, New York, NY, USA, 119–128.

[34] András Vargha and Harold D. Delaney. 2000. A Critique and Improvement of the "CL" Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (2000), pp. 101–132.

[35] W. Wen. 2012. Software fault localization based on program slicing spectrum. In *2012 34th International Conference on Software Engineering (ICSE)*. 1511–1514.

[36] W. E. Wong, V. Debroy, Y. Li, and R. Gao. 2012. Software Fault Localization Using DStar (D*). In *2012 IEEE Sixth International Conference on Software Security and Reliability*. 21–30. https://doi.org/10.1109/SERE.2012.12

[37] W. E. Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. 2016. A Survey on Software Fault Localization. *IEEE Transactions on Software Engineering* 42, 8 (August 2016), 707.

[38] W. Eric Wong, Yu Qi, Lei Zhao, and Kai-Yuan Cai. 2007. Effective Fault Localization using Code Coverage. In *Proceedings of the 31st Annual International Computer Software and Applications Conference - Volume 01 (COMPSAC '07)*. IEEE Computer Society, Washington, DC, USA, 449–456.

[39] Jifeng Xuan and M. Monperrus. 2014. Learning to Combine Multiple Ranking Metrics for Fault Localization. In *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME 2014)*. 191–200.

[40] Xin Ye, Razvan Bunescu, and Chang Liu. 2014. Learning to Rank Relevant Files for Bug Reports Using Domain Knowledge. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*. ACM, New York, NY, USA, 689–699. https://doi.org/10.1145/2635868.2635874

[41] Shin Yoo. 2012. Evolving Human Competitive Spectra-Based Fault Localisation Techniques. In *Search Based Software Engineering*, Gordon Fraser and Jerffeson Teixeira de Souza (Eds.). Lecture Notes in Computer Science, Vol. 7515. Springer Berlin Heidelberg, 244–258.

[42] Shin Yoo, Xiaoyuan Xie, Fei-Ching Kuo, Tsong Yueh Chen, and Mark Harman. 2017. Human Competitiveness of Genetic Programming in SBFL: Theoretical and Empirical Analysis. *ACM Transactions on Software Engineering and Methodology* 26, 1 (July 2017), 4:1–4:30.

[43] Mengshi Zhang, Xia Li, Lingming Zhang, and Sarfraz Khurshid. 2017. Boosting Spectrum-based Fault Localization Using PageRank. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2017)*. ACM, New York, NY, USA, 261–272.