Learning Fault Localisation for Both Humans and Machines using Multi-Objective GP

Kabdo Choi, Jeongju Sohn, and Shin Yoo

Korea Advanced Institute of Science and Technology,
Daejeon, Republic of Korea

Abstract. Genetic Programming has been successfully applied to fault localisation to learn ranking models that place the faulty program element as near the top as possible. However, it is also known that, when localisation results are used by Automatic Program Repair (APR) techniques, higher rankings of faulty program elements do not necessarily result in better repair effectiveness. Since APR techniques tend to use localisation scores as weights for program mutation, lower scores for non-faulty program elements are as important as high scores for faulty program elements. We formulate a multi-objective version of GP based fault localisation to learn ranking models that not only aim to place the faulty program element higher in the ranking, but also aim to assign as low scores as possible to non-faulty program elements. The results show minor improvements in the suspiciousness score distribution. However, surprisingly, the multi-objective formulation also results in more accurate fault localisation *ranking-wise*, placing 155 out of 386 faulty methods at the top, compared to 135 placed at the top by the single objective formulation.

Keywords: Fault Localisation, Multi-Objective Evolutionary Algorithm

1 Introduction

Genetic Programming has been successfully applied to fault localisation [9], initially to learn individual Spectrum-Based Fault Localisation (SBFL) risk evaluation formulæ [11] and subsequently to learn more complicated ranking models that take multiple SBFL formulæ as well as code and change metrics as input and produce rankings of program elements [7].

Increasingly, fault localisation techniques are being used by Automated Program Repair techniques, such as GenProg [8]: suspiciousness scores, i.e., the scores used for rankings, are often used as weights to determine parts of the program under repair that need to be patched. Noting this, Qi et al. evaluated fault localisation techniques using success rates of APR techniques as the effectiveness measure [6] and reported an interesting finding: SBFL formulas proven to produce better ranking than others [10] turned out to be less effective than formulæ they dominate when used with APR.

Inspired by Qi et al., we formulate a multi-objective learning to rank problem for fault localisation, aiming to evolve a ranking model that assigns not only higher scores (and, therefore, rankings) to faulty program elements, but also lower scores to non-faulty program elements. Thus, we aim to maintain better *rankings* for humans, while producing better *distributions* for APR techniques.

2 Locality Information Loss as Fitness Function

The primary fitness function used by the multi-objective version of FLUCCS is average ranking of the first faulty program elements computed against the faults in the training data. This section explains how we convert the distribution of suspiciousness scores into a secondary fitness function.

2.1 Locality Information Loss (LIL)

LIL is an evaluation metric for fault localization based on information theory [4]. Essentially, LIL treats the distribution of suspiciousness scores as a probability distribution, and computes the cross-entropy between the ground truth and the given score distribution using Kullback-Leibler divergence. The score distribution for the ground truth is give $\mathcal{L}(s_i) = 1$ if the program element s_i belongs to the set of faulty elements, S_f, and $0 < \epsilon \ll 1$ otherwise. LIL converts both the ground truth distribution, $\mathcal{L}$, and the given suspiciousness score distribution, τ, into probability distribution using linear normalisation, i.e. $P_\tau(s_i) = \frac{\tau(s_i)}{\sum_{i=1}^{n} \tau s_i}, (1 \leq i \leq n)$. Finally, LIL itself is computed as the Kullbeck-Leibler divergence between two distributions: $D_{KL}(P_\mathcal{L}||P_\tau) = \sum_i P_\mathcal{L}(s_i) \ln \frac{P_\mathcal{L}(s_i)}{P_\tau(s_i)}$.

2.2 Weighted Locality Information Loss

After initial investigation, we learnt that LIL in its basic form is not suitable as a fitness function. As the number of program elements grows, the faulty program elements, of which there are only a few, have decreasing impact on the final LIL score. Reducing the suspiciousness scores for all program elements becomes a more effective strategy for GP to learn, damaging the ranking based fitness.

To counter this, we introduce Weighted Locality Information Loss, wLIL, which is defined as $\sum_i w_i P_\mathcal{L}(s_i) \ln \frac{P_\mathcal{L}(s_i)}{P_\tau(s_i)}$. The value of the weight, w_i, depends on whether s_i is the faulty program element or not: $w_i = \frac{|S_f|}{|S|}$ if $s_i \in S_f$, and $\frac{|S \setminus S_f|}{|S|}$ if $s_i \notin S_f$ (S is the set of all program elements).

3 Experimental Setup

This section presents our research questions and the experimental set-up.

3.1 Research Questions

We investigate the following research questions to evaluate the effectiveness of the multi-objective version of FLUCCS, $\mathbb{F}_{MO}$, which uses NSGA-II [1] to implement multi-objective GP.

RQ1. Ranking Effectiveness: how effective is $\mathbb{F}_{MO}$ at ranking the faulty program elements higher than non-faulty elements?
RQ2. Distribution Effectiveness: how effective is $\mathbb{F}_{MO}$ at producing distributions of suspiciousness scores that resembles the ground truth?

Intuitively, RQ1 evaluates $\mathbb{F}_{MO}$ from the human perspective by checking the ranking of the faulty program elements, while RQ2 evaluates $\mathbb{F}_{MO}$ from the machine (i.e., APR) perspective. We use the original FLUCCS as the single objective baseline, $\mathbb{F}_{SO}$. To answer RQ1, we adopt the widely used evaluation metrics, $acc@n$ and wef, to compare $\mathbb{F}_{MO}$ and $\mathbb{F}_{SO}$: $acc@n$ counts the number of faults that have been ranked within the top n places by ranking models, whereas wef is the number of non-faulty program elements ranked higher than the first faulty program elements[1]. To answer RQ2, we measure the ratio between the highest suspiciousness scores of faulty and non-faulty program elements: casually, the higher the ratio is, the more *obvious* the faulty program elements appears to APR techniques. We report these ratios because wLIL values are hard to interpret intuitively.

3.2 Subject Programs

We use real world faults from `Defects4J` [3], the same benchmark that have been used in our previous works [7]. From the 395 faults provided by `Defects4J` 1.1.0 repository, we use 386 faults, excluding 9 faults that we could not reproduce in the method level localisation experiments. Table 1 contains the details of subject programs and faults.

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	131	30438–50523	4848–8880	2595–8443
Jfreechart	25	41075–51523	6578–8281	1586–2193
Mockito	36	2110–4385	747–1476	695–1399

3.3 Configuration

Both $\mathbb{F}_{MO}$ and $\mathbb{F}_{SO}$ are implemented using DEAP 1.2.2 [2], a Python evolutionary computation framework. We use tree-based GP, with single-point cross over with rate 1.0 and subtree mutation with rate 0.1. The population size is 40, and the maximum and minimum tree depth are eight and one respectively. The stopping criterion is after 100 generations. We use six GP operators: addition, subtraction, multiplication, safe division, negation, and safe square root. Both $\mathbb{F}_{MO}$ and $\mathbb{F}_{SO}$ use the same set of features and constant values as the previous work [7]. All experiments were performed on Ubuntu 16.04.4 LTS.

[1] Note that our primary fitness function is essentially the mean wef computed for faults in the training data-set.

To avoid overfitting, we adopt ten-fold cross validation: 386 faults have been divided into 10 folds, each consisting of 35 to 39 faults. Each fold is used as the test data set to validate the ranking models trained with the remaining folds. We repeat GP ten times for each fold: for $\mathbb{F}_{MO}$, from each run in a fold, we first choose the ranking model with the best ranking fitness on the final Pareto-front to represent the run. Subsequently, we choose the ranking models with median and minimum ranking fitness ($\mathbb{F}_{MO}^{med}$ and $\mathbb{F}_{MO}^{min}$) among the ten representatives. For $\mathbb{F}_{SO}$, we simply choose the best individual from each run in a fold as the representative, and subsequently choose ones with median and minimum ranking fitness ($\mathbb{F}_{SO}^{med}$ and $\mathbb{F}_{SO}^{min}$) among the ten representatives. Finally, all faults are localised by $\mathbb{F}_{MO}^{med}$, $\mathbb{F}_{MO}^{min}$, $\mathbb{F}_{SO}^{med}$, and $\mathbb{F}_{SO}^{min}$ trained in their corresponding folds.

Ranking models generated by both $\mathbb{F}_{MO}$ and $\mathbb{F}_{SO}$ are essentially a large expressions that produce suspiciousness scores. When raking program elements using these scores, it is possible for ties to take place. We use the maximum tie-breaking rule, which assigns the lowest rank to all of the tied elements.

Table 2. Ranking Effectiveness of Single and Multi-objective FLUCCS

Config.	Subject	Flt.	acc				wef		Config.	Subject	Flt.	acc				wef	
			@1	@3	@5	@10	mean	σ				@1	@3	@5	@10	mean	σ
$\mathbb{F}_{SO}^{med}$	Chart	25	15	18	20	22	6.6400	20.0497	$\mathbb{F}_{MO}^{med}$	Chart	25	17	20	23	24	1.3600	3.0447
	Clos.	131	34	66	81	92	29.9008	101.3193		Clos.	131	37	66	83	97	33.3282	118.3868
	Lang	63	27	44	49	54	2.8571	4.5595		Lang	63	36	48	54	61	2.5556	9.0374
	Math	105	42	56	58	70	46.2857	305.3330		Math	105	45	60	67	76	107.1143	674.0247
	Mock.	36	10	19	21	28	9.2500	22.5085		Mock.	36	9	17	21	28	15.5833	54.0896
	Time	26	7	13	15	19	133.6538	636.7294		Time	26	11	16	17	19	253.0385	854.3380
Overall		386	135	216	244	285	33.5000	239.1826	Overall		386	155	227	265	305	59.4508	426.7188
$\mathbb{F}_{SO}^{min}$	Chart	25	16	20	23	23	1.8400	4.3237	$\mathbb{F}_{MO}^{min}$	Chart	25	15	20	23	23	1.9600	4.7873
	Clos.	131	25	61	74	92	33.5191	113.6997		Clos.	131	38	66	79	95	31.6336	103.2980
	Lang	63	34	42	48	55	2.8571	4.4645		Lang	63	34	46	53	59	33.5555	247.6137
	Math	105	49	62	65	76	57.8571	454.5908		Math	105	39	55	60	68	64.9714	480.6590
	Mock.	36	9	17	22	29	10.0555	21.7496		Mock.	36	8	18	21	27	49.8611	212.5439
	Time	26	10	16	18	19	89.5769	379.9093		Time	26	8	17	18	20	142.1538	636.0536
Overall		386	143	218	250	294	34.6710	266.4820	Overall		386	142	222	254	292	48.2383	329.9654

4 Results

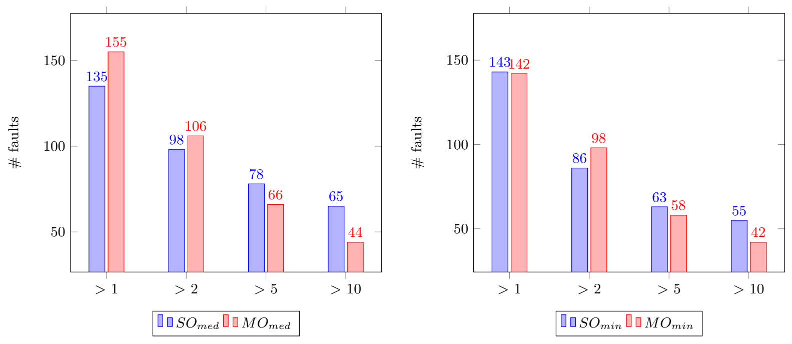
Table 2 shows the results of ranking models generated by $\mathbb{F}_{MO}$ and $\mathbb{F}_{SO}$. Median fitness models perform better, $\mathbb{F}_{SO}^{med}$ and $\mathbb{F}_{MO}^{med}$ localising 35% and 40% of the total faults at the top of ranking respectively. Both $\mathbb{F}_{SO}^{min}$ and $\mathbb{F}_{MO}^{min}$ places approximately 37% of the faults at the top in comparison. Within top 10, 73% and 76% of the faults are localized by $\mathbb{F}_{SO}^{med}$ and $\mathbb{F}_{SO}^{min}$, respectively; $\mathbb{F}_{MO}^{med}$ and $\mathbb{F}_{MO}^{min}$ place 79% and 75.6% within top 10.

Most notably, $\mathbb{F}_{MO}^{med}$ ranking models performs either better or almost equally well according to *acc*@1, when compared to $\mathbb{F}_{SO}^{med}$ counterparts. We interpret this as a similar phenomenon to that reported by Praditwong et al. [5] in software remodularisation: formulating the same problem in a multi-objective fashion contributes to better fitness than in the single objective formulation. While the results calls for a closer analysis, we cautiously posit that this improvement is due

Table 3. Effectiveness of Single and Multi-objective FLUCCS

Config.	Subject	Flt.	v_f/v_n				v_n/v_f				Config.	Subject	Flt.	v_f/v_n				v_n/v_f			
			> 1	> 2	> 5	> 10	≥ 1	≥ 2	≥ 5	≥ 10				> 1	> 2	> 5	> 10	≥ 1	≥ 2	≥ 5	≥ 10
$\mathbb{F}_{SO}^{med}$	Chart	25	15	12	12	6	10	7	3	3	$\mathbb{F}_{MO}^{med}$	Chart	25	17	13	11	4	8	7	6	4
	Clos.	131	34	24	17	15	97	74	58	50		Clos.	131	37	26	12	8	94	81	61	50
	Lang	63	27	24	21	20	36	23	19	15		Lang	63	36	29	20	18	27	19	11	7
	Math	105	42	25	20	19	63	45	30	26		Math	105	45	28	17	10	60	48	34	23
	Mock.	36	10	7	5	3	26	19	15	12		Mock.	36	9	5	2	2	27	22	12	10
	Time	26	7	6	3	2	19	11	4	4		Time	26	11	5	4	2	15	13	9	9
Overall		386	135	98	78	65	251	179	129	110	Overall		386	155	106	66	44	231	190	133	103
$\mathbb{F}_{SO}^{min}$	Chart	25	16	13	11	8	9	6	6	6	$\mathbb{F}_{MO}^{min}$	Chart	25	15	9	5	2	10	8	7	2
	Clos.	131	25	18	14	14	106	64	51	45		Clos.	131	38	28	13	10	93	75	60	49
	Lang	63	34	16	15	13	29	20	19	16		Lang	63	34	26	22	17	29	19	10	8
	Math	105	49	27	18	15	56	41	32	28		Math	105	39	23	13	8	66	49	37	25
	Mock.	36	9	5	2	2	27	13	10	7		Mock.	36	8	6	1	1	28	22	14	9
	Time	26	10	7	3	3	16	10	10	9		Time	26	8	6	4	4	18	15	10	8
Overall		386	143	86	63	55	243	154	128	111	Overall		386	142	98	58	42	244	188	138	101

Fig. 1. Histograms of $\frac{v_f}{v_n}$ ratios achieved by $\mathbb{F}_{MO}$ and $\mathbb{F}_{SO}$



to the increased diversity during the multi-objective evolution. Interestingly, in terms of wef, $\mathbb{F}_{SO}$ tends to outperform $\mathbb{F}_{MO}$, which is as expected because $\mathbb{F}_{SO}$ can focus on improving wef alone whereas $\mathbb{F}_{MO}$ has to maintain Pareto-optimal populations. To answer RQ1: $\mathbb{F}_{MO}$ can rank as effectively as $\mathbb{F}_{SO}$.

To evaluate the distribution effectiveness, we report the ratio between the maximum score among faulty elements, v_f, and the maximum score among non-faulty elements, v_n. For both ratios $\frac{v_f}{v_n}$ and $\frac{v_n}{v_f}$, we count the number of faults for which the ratio exceeded $n = 1, 2, 5, 10$. The results are shown in Table 3 and Figure 1. $\mathbb{F}_{MO}$ localise more faults with higher ratios up to $n = 2$, but fail to localise more faults with ratios higher than five. However, also note that the number of faults whose $\frac{v_n}{v_f}$ is greater than 10, i.e., the number of faults that are extremely difficult to localise, has been decreased by $\mathbb{F}_{MO}$ ranking models: from 110 to 103 by $\mathbb{F}_{MO}^{med}$, and from 111 to 101 by $\mathbb{F}_{MO}^{min}$, respectively. We suspect that the secondary objective, wLIL, encouraged the faulty program elements to be assigned with higher scores. To answer RQ2: $\mathbb{F}_{MO}$ does produce better distributions, but its effect is limited.

5 Conclusion

We report the first attempt to evolve ranking models for fault localisation that is useful for both humans and machines using Multi-Objective GP. The results suggest that the added diversity produces better rankings.

Acknowledgements This research was supported by the Korean MSIT(Ministry of Science and ICT), under the National Program for Excellence in SW (2016-0-00018), supervised by the IITP(Institute for Information & communications Technology Promotion).

References

1. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Transactions on Evolutionary Computation 6(2), 182–197 (Apr 2002)
2. Fortin, F.A., De Rainville, F.M., Gardner, M.A.G., Parizeau, M., Gagné, C.: Deap: Evolutionary algorithms made easy. J. Mach. Learn. Res. 13(1), 2171–2175 (Jul 2012)
3. Just, R., Jalali, D., Ernst, M.D.: 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. pp. 437–440. ISSTA 2014, ACM, New York, NY, USA (2014)
4. Moon, S., Kim, Y., Kim, M., Yoo, S.: Ask the mutants: Mutating faulty programs for fault localization. In: Proceedings of the 2014 IEEE International Conference on Software Testing, Verification, and Validation. pp. 153–162. ICST '14, IEEE Computer Society, Washington, DC, USA (2014)
5. Praditwong, K., Harman, M., Yao, X.: Software module clustering as a multi-objective search problem. IEEE Transactions on Software Engineering 37(2), 264–282 (March-April 2010)
6. Qi, Y., Mao, X., Lei, Y., Wang, C.: Using automated program repair for evaluating the effectiveness of fault localization techniques. In: Proceedings of the 2013 International Symposium on Software Testing and Analysis. pp. 191–201. ISSTA 2013, ACM, New York, NY, USA (2013)
7. Sohn, J., Yoo, S.: FLUCCS: Using code and change metrics to improve fault localization. In: Proceedings of the 26th International Symposium on Software Testing and Analysis. pp. 273–283. ISSTA 2017, ACM (2017)
8. Weimer, W., Nguyen, T., Goues, C.L., Forrest, S.: Automatically finding patches using genetic programming. In: Proceedings of the 31st IEEE International Conference on Software Engineering (ICSE '09). pp. 364–374 (16-24 May 2009)
9. Wong, W.E., Gao, R., Li, Y., Abreu, R., Wotawa, F.: A survey on software fault localization. IEEE Transactions on Software Engineering 42(8), 707 (August 2016)
10. Xie, X., Chen, T.Y., Kuo, F.C., Xu, B.: A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. ACM Transactions on Software Engineering Methodology 22(4), 31:1–31:40 (October 2013)
11. Yoo, S.: Evolving human competitive spectra-based fault localisation techniques. In: Fraser, G., Teixeira de Souza, J. (eds.) Search Based Software Engineering, Lecture Notes in Computer Science, vol. 7515, pp. 244–258. Springer (2012)