Chapter 8
Spectrum-Based Fault Localization for Multiple Faults

Abstract Currently, SBFL for multi-fault scenario has received more and more attention. In general, there are two common ways to perform multiple fault localization, namely, sequential debugging (SD) and parallel debugging (PD). SD executes the program against all failed test cases and all passed test cases in test suite and achieves the goal of eliminating all faults through localizing one fault at a time iteratively. PD separates failed test cases and forms a number of fault-focused clusters (each cluster is aimed at one fault); each fault-focused cluster merged with passed cases can be executed simultaneously for parallel fault localization. Because of the lower cost and higher efficiency, PD is more widely investigated. This chapter will introduce two important techniques of PD, namely, P2 and MSeer.

8.1 Challenge in SBFL: Dealing with Multiple Faults

A lot of studies around spectrum-based fault localization (SBFL) have been carried out by a significant number of researchers in recent years, but many of them are based on single-fault environment, that is, assuming that there is only one fault in the program, which is obviously inconsistent with reality. In single-fault environment, all failed executions are unquestionably caused by one fault in the program; however, in multiple-fault environment, various failed executions have different origins, i.e., the due-to relationship between a failure and its corresponding fault(s) is not defined in advance, which would decrease the effectiveness of the fault localization technique.

In general, there are two common ways to solve the problem of multiple-fault localization: sequential debugging (SD) and parallel debugging (PD). SD executes the program PG against all failed test cases and all passed test cases in test suite TS and achieves the goal of eliminating all faults through localizing one fault at a time iteratively. PD separates failed test cases and forms a number of fault-focused clusters (each cluster is aimed at one fault); each fault-focused cluster merged with passed cases can be executed simultaneously for parallel fault localization.

X. Xie, B. Xu, *Essential Spectrum-based Fault Localization*,
https://doi.org/10.1007/978-981-33-6179-9_8

8.2 Sequential Debugging

Sequential debugging is also known as One-Bug-at-A-time (OBA) method. It is one of the easiest methods for solving software multiple-fault localization. OBA starts working when the failures of PG are observed through a given TS. After localizing and fixing the first fault according to the ranking, the debugger will continue to execute fixed PG against TS and observe the execution result of TS and then re-localize and move faults again until no execution failure has been detected. Figure 8.1 demonstrates the OBA process using SBFL technique.

The detailed steps for OBA using SBFL are as follows:

- Step 1: Execute PG against TS and collect spectrum information simultaneously;
- Step 2: Input the spectrum information into the risk evaluation formula, and then generate the statement risk value ranking;
- Step 3: Localize and fix one fault according to the ranking;
- Step 4: Repeat Step 1 - Step 3 until no execution failure can be observed.

It is obvious that One-Bug-at-a-time is actually a sequential debugging method that starts the next localization process only after the current fault has been fixed. Its evident drawbacks are high costs of time and low efficiency. In addition, OBA is often used for fault localization based on Bayesian reasoning. However, Wong et al. pointed out that this is equivalent to assuming that multiple faults existing in the program are independent of one another [12]; in other words, the possible correlation between the various faults is disregarded, which is certainly incompatible with the actual situation. Debroy and Wong investigated the interaction between multiple faults and found that there were mainly two conditions: destructive interference and constructive interference [2]. The former refers to the execution failure, which is triggered when a fault exists on its own, while the execution result becomes correct when another fault is injected. The latter means that if there is a certain fault on its own, the execution failure will not be triggered; only if there are two faults at the same time, the program will show abnormality. The conclusion

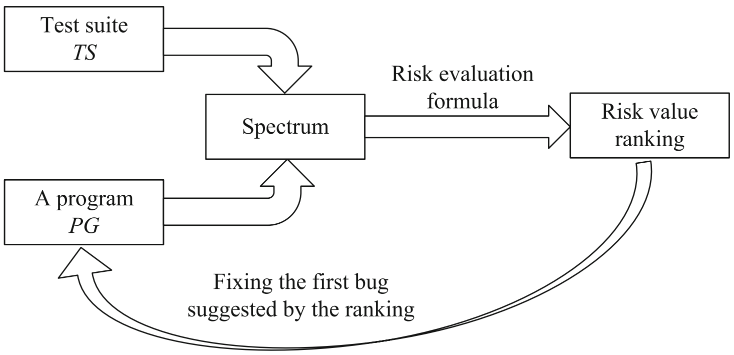

Fig. 8.1 An overview of OBA using SBFL technique

in [2] confirms that destructive interference is more common than constructive interference.

In summary, OBA not only has the disadvantages of high cost and low efficiency, but is also often inconsistent with the practice. In order to overcome these drawbacks of sequential debugging, many researchers are focusing on parallel debugging and proposing a series of new methods.

8.3 Parallel Debugging

The basic principle of multiple-fault parallel localization is to separate failed cases in TS according to the similarity measure, so that the failed cases in the same cluster are triggered by the same fault, and different faults cause the failed cases between different clusters. That is, the multiple-fault is isolated into multiple single-faults by generating a number of $fault\text{-}focused$ clusters, which reduces the complexity of the problem.

Clustering is a recognized scheme for separating failed test cases in TS. A proper representation of failed cases is the prerequisite for a high-quality clustering process. There are mainly two types of vectors widely used to represent failed test cases. One is the path of execution coverage, which is a binary vector transformed from the trace of execution of a failed test case on PG (similar to the representation used to compute T-proximity). Another is the risk value ranking, which is a statement risk value list generated by the risk evaluation formula using spectrum information gathered from a failed case and all passed cases on PG (similar to the representation used to compute R-proximity). Liu et al. have proved that the latter has higher efficiency than the former [7], because the fault could trigger execution failure in various ways. In other words, even failed cases caused by the same fault can have different paths of execution, which is obviously ignored by the representation using execution path coverage.

Several researchers have proposed a series of SBFL techniques for multiple-fault based on R-proximity accordingly, such as P2 proposed by Jones et al. [5] and MSeer proposed by Gao et al. [3].

8.3.1 Approach: P2

In [5], two parallel debugging techniques for multiple-fault were proposed: P1 based on behavior models and fault localizing results and P2 based on fault localizing results only. However, [1, 3, 4] all pointed out that Jones et al. did not provide technical details for the clustering process of P1 in [5], so P1 was hard to reproduce. In contrast, P2 has been used by many researchers for experimental comparison. The detail of P2 is shown in Fig. 8.2.

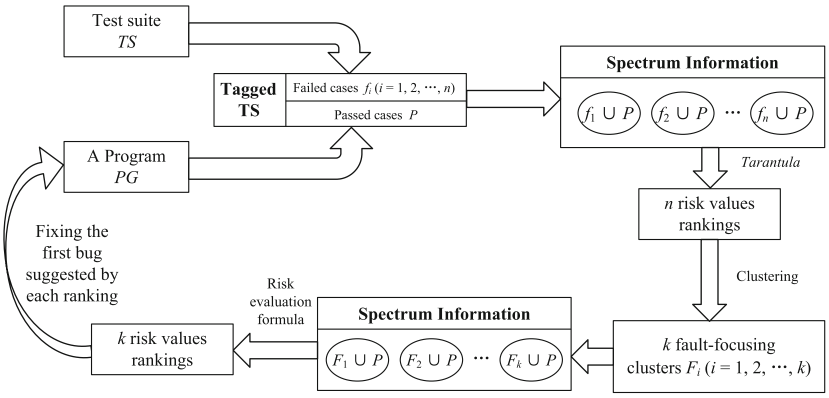

Fig. 8.2 An overview of P2

The detailed steps for P2 are as follows:

- Step 1: Execute PG against TS and gather coverage information, and then divide the test cases in TS into failed cases and passed cases by comparing between expected outputs and target outputs;
- Step 2: Merge n failed cases with all passed cases to obtain n sub-TSs;
- Step 3: Input spectrum information of n sub-TSs into $Tarantula$ to generate n rankings;
- Step 4: Conduct hierarchical clustering on n rankings according to Jaccard distance to generate k fault-focused clusters;
- Step 5: Merge the failed cases in k clusters with all passed cases, respectively, to obtain k fault-focused TSs;
- Step 6: Input the spectrum information of these fault-focused TSs into the risk evaluation formula to generate k rankings;
- Step 7: Localize and fix the first fault suggested by each ranking;
- Step 8: Repeat Step 1 - Step7 until there is no detectable execution failure.

The distance measure and clustering algorithm in Step 4 is the core of P2 and will directly affect the effectiveness of this technique. The following describes the Jaccard distance metric and hierarchical clustering algorithm involved in P2.

For rankings r_i and r_j that represent two failed test cases, Jaccard computes the distance (between 0 and 1) by taking the ratio of the intersection of the two rankings to the union.

$$Jaccard(r_i, r_j) = 1 - \frac{|r_i \cap r_j|}{|r_i \cup r_j|}$$

where $|r_i \cap r_j|$ and $|r_i \cup r_j|$ are the size of the intersection and the union of r_i and r_j, respectively. To determine whether the relationship between two rankings is $similar$ or $not\ similar$, P2 sets the $threshold$ = 0.5; r_i and r_j are not considered to be *similar* unless the distance between them is smaller than the *threshold*. It should be noted that when measuring Jaccard distance, P2 only considers the top *MostSusp* part of the rankings and the rest are deemed *not of interest*.

Jones et al. first presented the definition of *Expense* to evaluate the effectiveness of P2.

$$Expense = \frac{rank\ of\ fault}{size\ of\ program} \times 100\%$$

Based on *Expense*, Jones et al. further proposed two metrics, *total developer expense* (denote as D) and *critical expense to a failure-free program* (denote as FF), to conduct a more comprehensive evaluation of P2.

$$D = \sum_{i=1}^{|faults|} Expense_i$$

$$FF = \sum_{i=1}^{|iterations|} max\{Expense_f | f \text{ is a subtask at iteration } i\}$$

D is used to evaluate total costs paid by all debuggers to localize all faults in the program in a sequential or parallel manner and captures essential quantities such as employee-hours and payroll-expense; FF is used to evaluate the cost to deliver a failure-free program and can be used to calculate the time required to complete the multiple-fault localization task. Jones et al. also pointed out that the failure-free program does not mean the program has no fault, only that it does not present abnormalities against the current TS.

8.3.2 Approach: MSeer

Gao et al., who are also interested in SBFL for multiple faults, argue that P2 has two apparent disadvantages. Firstly, all statements are assigned the same weight by Jaccard distance metric; in other words, no attention is paid to enhancing the contribution of high-risk value statements when measuring the distance between rankings, so that the distance between rankings cannot be measured accurately. Secondly, the hierarchical clustering algorithm does not have strong effectiveness in the process of multiple-fault localization. Thus, Gao et al. proposed a parallel debugging method [3], MSeer, which is shown in Fig. 8.3.

There are three fundamental differences in MSeer compared with P2: the use of Crosstab in ranking generation, the revised Kendall tau metric in distance metric, and the K-medoids algorithm based on the estimated number of clusters and the initial medoids.

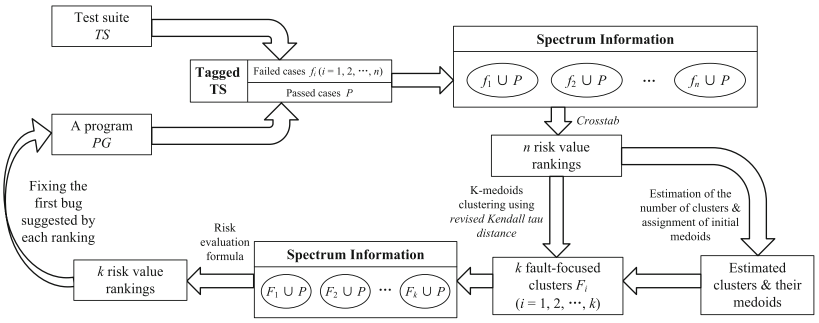
Test suite TS
Tagged TS
Failed cases f_i (i = 1, 2, ···, n)
Passed cases P
Spectrum Information
$f_1 \cup P$
$f_2 \cup P$
$f_n \cup P$
A program PG
Crosstab
n risk value rankings
Fixing the first bug suggested by each ranking
K-medoids clustering using revised Kendall tau distance
Estimation of the number of clusters & assignment of initial medoids
Risk evaluation formula
Spectrum Information
$F_1 \cup P$
$F_2 \cup P$
$F_k \cup P$
k risk value rankings
k fault-focused clusters F_i (i = 1, 2, ···, k)
Estimated clusters & their medoids

Fig. 8.3 An overview of MSeer

(1) **Crosstab: A risk evaluation formula.** Crosstab is a risk evaluation technique proposed by Wong et al. [11], which uses spectrum information to construct a crosstab for each statement by computing the chi-square statistic and the coefficient of contingency [8]. Crosstab is given by

$$\chi^2 = \frac{(a_{ef} - E_{ef})^2}{E_{ef}} + \frac{(a_{ep} - E_{ep})^2}{E_{ep}} + \frac{(a_{nf} - E_{nf})^2}{E_{nf}} + \frac{(a_{np} - E_{np})^2}{E_{np}}$$

Crosstab will first calculate φ for each statement to determine its association with failed and passed execution and then use φ to decide if the statement should be assigned χ^2, -χ^2, or 0. Please refer to [11] for more details on this technique.

(2) **Revised Kendall tau distance.** By counting the number of discordant pairs between two rankings of the same size [6], Kendall tau computes distance between them directly. Gao et al. pointed out that although Kendall tau distance has been successfully used in other research such as information retrieval [10] and bioengineering [9], it cannot be directly applied to SBFL since it assigns identical weight to all statements. Kendall tau distance metric is given by

$$D(x, y) = \sum_{1 \le i < j \le m} K\left(s_i, s_j\right)$$

where x and y are two rankings to be compared and m is the number of statements included in each ranking. In the original Kendall tau, $K(s_i, s_j)$ is defined as follows:

- If $(x(s_i) - x(s_j)) \times (y(s_i) - y(s_j)) < 0$, $K(s_i, s_j) = 1$;
- Otherwise, $K(s_i, s_j) = 0$

where $x(s_i)$, $x(s_j)$, $y(s_i)$, $y(s_j)$ represent the positions of statement s_i and statement s_j in ranking x and ranking y, respectively. Thus, even though s_i and s_j are at a very low position in the ranking, their contribution to the distance would still be the same as that of the statement at a high position in the ranking, which is contrary to Gao et al.'s viewpoint. To solve this problem, Gao et al. proposed a revised Kendall tau distance:

$$D'(x, y) = \sum_{1 \le i < j \le N} K'\left(s_i, s_j\right)$$

where x and y are two rankings to be compared and N is the number of statements included in each ranking. In the revised Kendall tau, $K'(s_i, s_j)$ is defined as follows:

- If $(x(s_i) - x(s_j)) \times (y(s_i) - y(s_j)) < 0$, $K'(s_i, s_j) = x(s_i)^{-1} + x(s_j)^{-1} + y(s_i)^{-1} + y(s_j)^{-1}$;
- Otherwise, $K'(s_i, s_j) = 0$

Obviously, the greater is the risk value of s_i and s_j, the higher the value of $K'(s_i, s_j)$, which met Gao et al.'s preceding concept.

(3) **Estimation of the number of clusters and assignment of initial medoids.** As a mainstream clustering algorithm, K-medoids is applied in many fields including software fault localization. One limitation of K-medoids is that the number of clusters must be determined in advance. Gao et al. argue that in software multiple-fault localization, the number of clusters generated by clustering is expected to be the same as the number of faults in PG; however, the latter is often not known by the debugger, which limits the application of K-medoids in SBFL for multiple-fault. On the other hand, Gao et al. also mention that K-medoids has another obvious drawback, i.e., in the process of minimizing the cost function, K-medoids has to examine each individual possible combination of rankings as initial medoids, which will make the cost significantly higher. To eliminate this limitation, Gao et al. proposed an approach for estimating the number of clusters and determining the initial medoids simultaneously for K-medoids [3], as mentioned below.

Execute PG against TS, supposing there are n failed test cases in TS. Merge these n failed cases and all passed cases P respectively to form n sub-TSs, then input sub-TSs into Crosstab to generate n rankings$\{r_1, r_2, r_3, \ldots, r_n\}$.

- Step 1: Compute the revised Kendall tau distance between r_i and r_j $(1 \le i, j \le n)$;
- Step 2: Assign the potential value P_i^0 for each ranking r_i $(1 \le i \le n)$;

$$P_i^0 = \sum_{j=1}^{n} \mathrm{e}^{-\alpha D'\left(r_i, r_j\right)^2}$$

where $\alpha = 4 / \psi^2$ and ψ equals to half of the 5 percent winsorized mean of the distance between two distinct rankings.

- Step 3: Once P_i^θ is computed, select the ranking with the highest potential value as R^θ, and set its potential value as M^θ (randomly choose one when multiple rankings share the same highest potential value). Set R^0 as the first cluster medoid and then proceed to Step 4. The criterion for determining the number of clusters is provided by the algorithm in Algorithm 5.[1]

Algorithm 5: A criterion to determine the number of clusters [3]

```
if $M^\theta > \overline{\varepsilon} M^0$ ($\overline{\varepsilon} = 0.5$ and $\underline{\varepsilon} = 0.15$)
  Accept $R^\theta$ as a cluster medoid and go to Step 4
else if $M^\theta < \underline{\varepsilon} M^0$
  Reject $R^\theta$ and stop
else
  Let $D'_{min}$ = [shortest of the revisited Kendall-tau distance between $R^\theta$ and all previously found cluster medoids]
  if $\frac{D'_{min}}{\psi} + \frac{M^\theta}{M^0} \geq 1$
    Accept $R^\theta$ as a cluster medoid and go to Step 4
  else
    (1) Reject $R^\theta$ and set the potential value of $M^\theta$ to 0
    (2) Select the ranking with the next highest potential value as $R^\theta$ and assign its potential value as the new $M^\theta$
    (3) Repeat the stopping criterion from the beginning
  end if
end if
```

- Step 4: Update the potential value of each ranking, and then go back to Step 3.

$$P_i^{\theta+1} = P_i^\theta - M^\theta \times e^{-\beta D'\left(r_i, R^\theta\right)^2}$$

where $\beta = 16 / 9\psi^2$.

It follows that MSeer primarily improves Step 3 and Step 4 of P2. The detailed process of MSeer is not described here due to the remaining steps of these two techniques being essentially identical.

References

1. Abreu R, Zoeteweij P, Gemund AJV (2009) Spectrum-based multiple fault localization. In: Proceedings of the 24th IEEE/ACM international conference on automated software engineering, pp 88–99. https://doi.org/10.1109/ASE.2009.25

[1]©2019 IEEE. Reprinted, with permission from IEEE Transactions on Software Engineering; November, 2019. Vol. 45, No. 3, 301–318. DOI: https://doi.org/10.1109/TSE.2017.2776912 (Ref. [3]).

2. Debroy V, Wong WE (2009) Insights on fault interference for programs with multiple bugs. In: Proceedings of the 20th international symposium on software reliability engineering, pp 165–174. https://doi.org/10.1109/ISSRE.2009.14
3. Gao R, Wong WE (2019) Mseer-an advanced technique for locating multiple bugs in parallel. IEEE Trans Softw Eng 45(3):301–318. https://doi.org/10.1109/TSE.2017.2776912
4. Hogerle W, Steimann F, Frenkel M (2014) More debugging in parallel. In: Proceedings of the 25th international symposium on software reliability engineering, pp 133–143. https://doi.org/10.1109/ISSRE.2014.29
5. Jones JA, Bowring JF, Harrold MJ (2007) Debugging in parallel. In: Proceedings of ACM SIGSOFT international symposium on software testing and analysis, pp 16–26. https://doi.org/10.1145/1273463.1273468
6. Kendall MG, Gibbons J (1990) Rank correlation method. Biometrika 11 en 12. https://doi.org/10.2307/2333282
7. Liu C, Zhang X, Han J (2008) A systematic study of failure proximity. IEEE Trans Softw Eng 34(6):826–843. https://doi.org/10.1109/TSE.2008.66
8. Sahoo S, Criswell J, Geigle C, Adve V (2013) Using likely invariants for automated software fault localization. ACM SIGPLAN Not 48(4):139–152. https://doi.org/10.1145/2490301.2451131
9. Sengupta D, Bandyopadhyay S, Maulik U (2010) A novel measure for evaluating an ordered list: application in microrna target prediction. In: Proceedings of the international symposium on biocomputing, pp 1–7. https://doi.org/10.1145/1722024.1722067
10. Teevan J, Dumais S, Horvitz E (2007) Characterizing the value of personalizing search. In: Proceedings of the 30th international ACM sigir conference on research & development in information retrieval, pp 757–758. https://doi.org/10.1145/1277741.1277894
11. Wong WE, Debroy V, Xu D (2012) Towards better fault localization: a crosstab-based statistical approach. IEEE Trans Syst Man Cybern 42(3):378–396. https://doi.org/10.1109/TSMCC.2011.2118751
12. Wong WE, Gao R, Li Y, Abreu R, Wotawa F (2016) A survey on software fault localization. IEEE Trans Softw Eng 42(8):707–740. https://doi.org/10.1109/TSE.2016.2521368