
FRONTE: Finding Bug Inducing Commits from Failures

Gabin An

School of Computing, KAIST
Daejeon, Republic of Korea
agb94@kaist.ac.kr

Jingun Hong

SAP Labs Korea
Seoul, Republic of Korea
jingun.hong@sap.com

Naryeong Kim

School of Computing, KAIST
Daejeon, Republic of Korea
kimnal1234@kaist.ac.kr

Shin Yoo

School of Computing, KAIST
Daejeon, Republic of Korea
shin.yoo@kaist.ac.kr

In Proceedings of 20th International Conference on Software Engineering(ICSE), 2023, pp. 589–601

BIC（バグ混入コミット）

| プログラムに，バグのあるコードを導入するコミット

| BICを正確に特定するメリット

- BIC以前のリビジョンに戻すことがバグ修正において有効^{[2][3]}
- バグ混入原因の理解^{[1][7]}
- バグ修正タスクの割り当て^[6]

デバッグ時に正確にBICを特定することが
効率的なバグ修正，バグトリアージ^[6]にも繋がる

- [2] R. Wu, M. Wen, S.-C. Cheung, and H. Zhang, “ChangeLocator: locate crash-inducing changes based on crash reports,” *Empirical Software Engineering*, vol. 23, no. 5, pp. 2866–2900, nov 2017. [Online].
- [3] M. Wen, Y. Liu, and S.-C. Cheung, “Boosting automated program repair with bug-inducing commits,” in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*. ACM, jun 2020. [Online].
- [6] V. Murali, L. Gross, R. Qian, and S. Chandra, “Industry-scale IR-based bug localization: A perspective from facebook,” in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, may 2021. [Online].
- [1] J. Śliwowski, T. Zimmermann, and A. Zeller, “When do changes induce fixes?” *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1–5, jul 2005. [Online].
- [7] G. Rodríguez-Pérez, G. Robles, A. Serebrenik, A. Zaidman, D. M. Germán, and J. M. Gonzalez-Barahona, “How bugs are born: a model to identify how bugs are introduced in software components,” *Empirical Software Engineering*, vol. 25, no. 2, pp. 1294–1340, feb 2020. [Online].

BIC特定の既存研究

| 2分探索 [9]

- コミット履歴の各スナップショットがバグかどうかを探索

| SZZ^[1]

- バグ修正コミットやメタデータからBICを特定
- バグ修正後には適用可能だが、デバッグ時には適用不可

| IRベース^{[5][10]}

- バグレポートとソースコードとのテキスト類似度を計算
- バグ報告前には適用不可・バグレポートの品質に依存

[5] M. Wen, R. Wu, and S.-C. Cheung, “Locus: locating bugs from software changes,” in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ACM, aug 2016. [Online].

[9] “Git-bisect-lk2009 documentation.” [Online].

[10] R. Bhagwan, R. Kumar, C. S. Maddila, and A. A. Philip, “Orca: Differential bug localization in large-scale services,” in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 493–509.

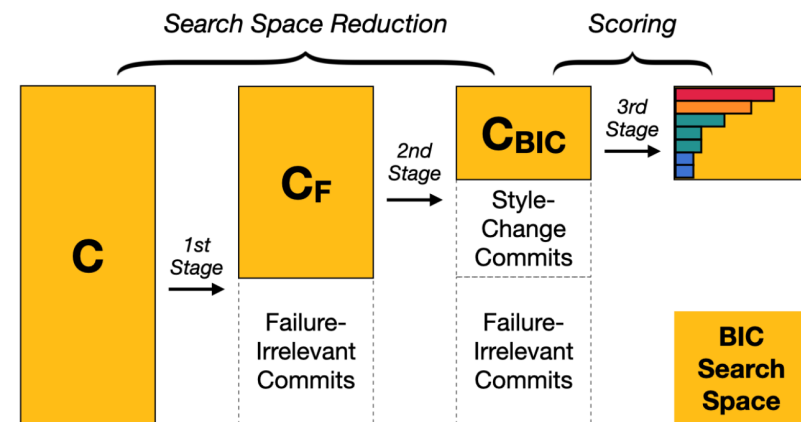
提案手法 | 概要

| 目的

- 正確, 効率的, デバッグ時に利用可能なBIC特定手法の提案
 - テストカバレッジとコミット履歴を使用

| FONTE

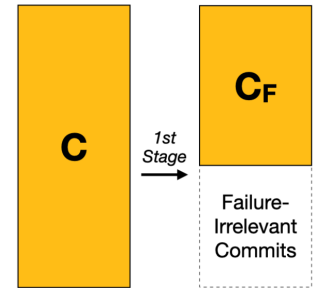
- テストカバレッジを用いて無関係なコミットを除外
- FL(Fault Localization)とコミット履歴からBICスコアを割当て



FRONTE | Stage1 & Stage2

| Stage 1 : 失敗関連コミットのフィルタリング

- 失敗テストケースのカバレッジから不審なコード要素の特定
- BIC候補の探索 : $E_F = \bigcup_{t \in T_F} \{e \in E | (t, e) \in \text{Cover}\}$
- 検索空間の削減 : $C_F = \bigcup_{e \in E_F} \{c \in C | (c, e) \in \text{Evolve}\}$

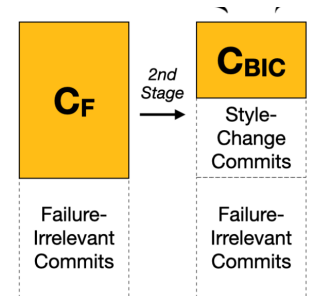


| Stage 2 : スタイル変更コミットのフィルタリング^[17]

- コミット前後のソースコードのASTを比較

```
14 -           if (escapeSingleQuote) out.write
      ('\\');
15 +           if (escapeSingleQuote) {
16 +               out.write('\\');
17 +           }
```

$$E_F^c = \{e \in E_F | (c, e) \in \text{Evolve}\}$$



[17] S. Kim, T. Zimmermann, K. Pan, and E. J. J. Whitehead, "Automatic identification of bug-introducing changes," in *21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*. IEEE, sep 2006. [Online].

FRONTE | Stage3

| Stage 3 : FLスコアとコミット履歴を用いた順位づけ

- 検索空間からFL手法^[11]でコードレベルの不審度を順位づけ
- 投票ベースでFLスコア順位をコミットレベルの順位に反映^{[19]-[21]}

$$vote(e) = \frac{\alpha * susp(e) + (1 - \alpha) * 1}{rank_{\tau}(e)}$$

Code Element		e_1	e_2	e_3	e_4	e_5
Score		1.0	0.6	0.6	0.6	0.3
$rank_{max}$		1	4	4	4	5
$rank_{dense}$		1	2	2	2	3
vote	$\alpha = 0, \tau = max$	1.00	0.25	0.25	0.25	0.20
	$\alpha = 1, \tau = max$	1.00	0.15	0.15	0.15	0.06
	$\alpha = 0, \tau = dense$	1.00	0.50	0.50	0.50	0.33
	$\alpha = 1, \tau = dense$	1.00	0.30	0.30	0.30	0.10

- 時間による減衰重み : 古いコミットほど投票力を減少
 - 古いコミットはすでに報告・修正されている^[2]

$$depth(e, c) = |\{c' \in C_{BIC} | (c', e) \in \text{Evolve} \wedge c'.time > c.time\}|$$

[11] W.E.Wong, R.Gao, Y.Li, R.Abreu, and F.Wotawa, "A survey on software fault localization," *IEEE Transactions on Software Engineering*, vol.42,no 8, pp.707–740, aug 2016.

[19] J. Sohn and S. Yoo, "Why train-and-select when you can use them all?" in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, jul 2019. [Online].

[20] J. Sohn, G. An, J. Hong, D. Hwang, and S. Yoo, "Assisting bug report assignment using automated fault localisation: An industrial case study," in *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, apr 2021. [Online].

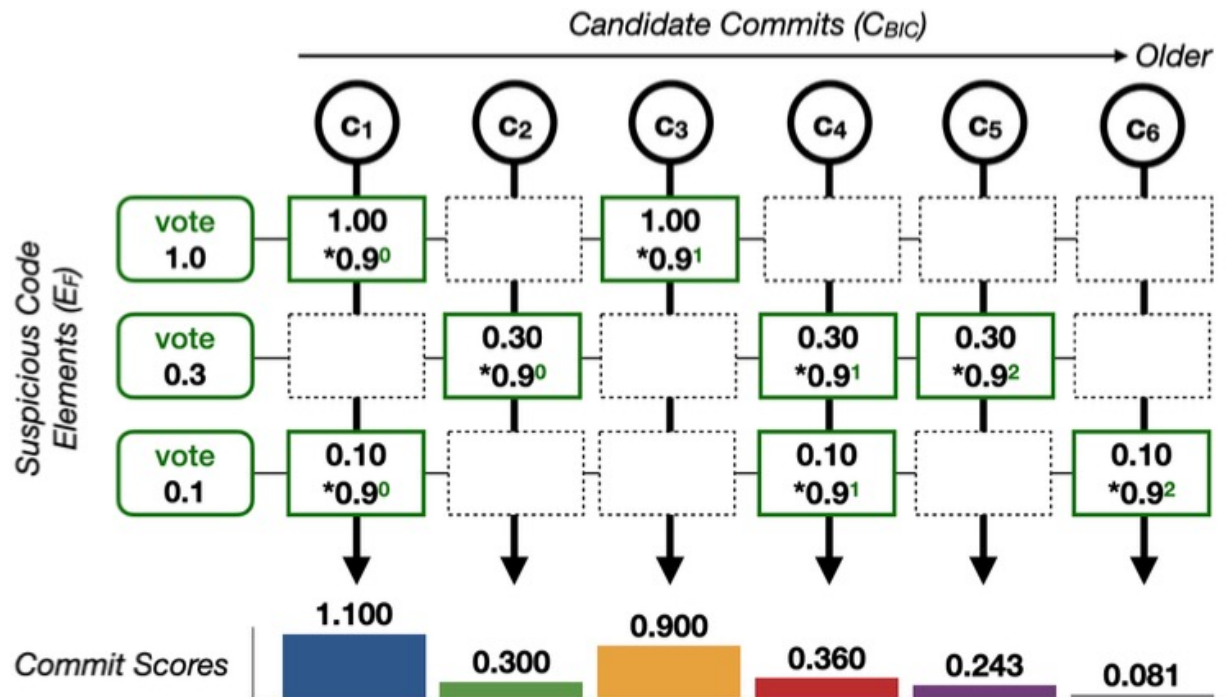
[21] S. Habchi, G. Haben, J. Sohn, A. Franci, M. Papadakis, M. Cordy, and Y. Le Traon, "What made this test flake? pinpointing classes responsible for test flakiness," *arXiv e-prints*, pp. arXiv–2207, 2022. [Online].

FRONTE | Stage3

| Stage 3 : バグ限局とコミット履歴を用いた順位づけ

$$\text{commitScore}(c) = \sum_{e \in E_F^c} \text{vote}(e) * (1 - \lambda)^{\text{depth}(e,c)}$$

ー 計算例



FRONTE | Stage3

| Stage 3 : バグ限局とコミット履歴を用いた順位づけ

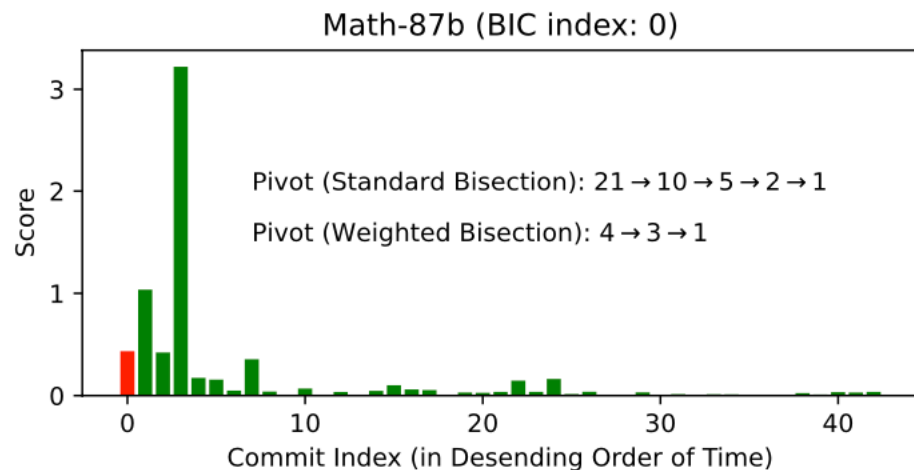
– 最終的な順位の算出 : 2分探索で検索空間を絞り込み

$$s(c) = \begin{cases} \text{commitScore}(c) & \text{if } c \in C_{BIC} \\ 0 & \text{otherwise} \end{cases}$$

プログラムが大規模
バグ検出テストに時間がかかる



デバッグパイプライン全体が滞る



探索効率化のために検索ピポットは
スコアの数を半分にするコミット

RQ

| RQ1

- FONTEはどの程度正確にBICを順位付けできるか

| RQ2

- スコアの数で2分探索は、順位で2分探索する場合と比較してどの程度効率的か

| RQ3

- FL手法はFONTEの性能にどのような影響を与えるか

データセット

| Wenら[4]のデータセット

- Defect4Jから91個のデータを選択

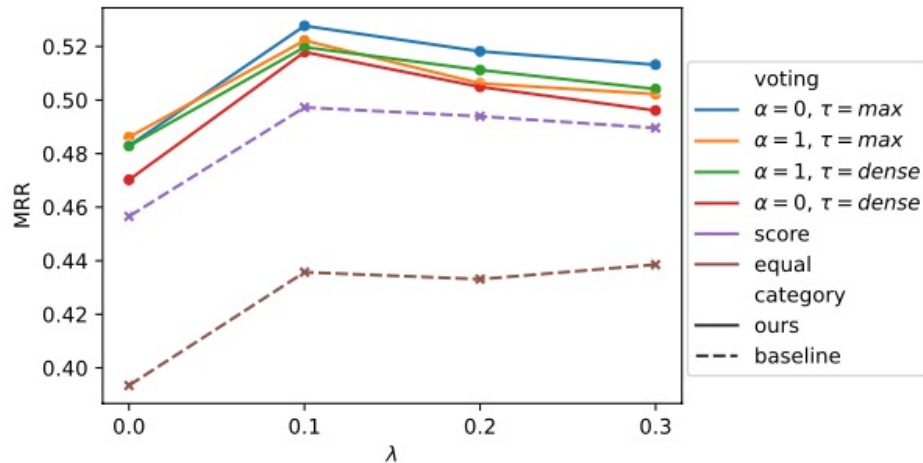
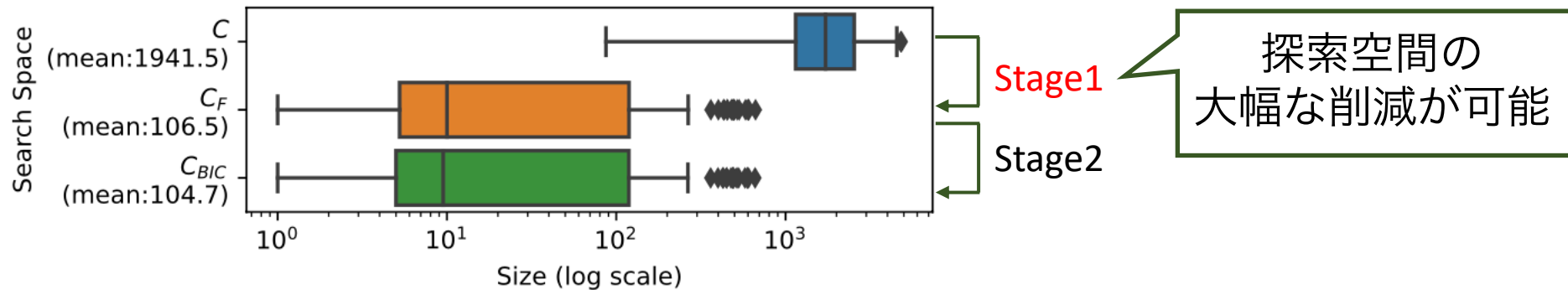
| フィルタリング

- 24データは、データが不完全/本分析で使用できない

| 本実験のデータセット

- 130個のデータ
- 67データ + 63データ（手作業で新たに収集）

RQ1 | FONTEはどの程度正確にBICを順位付けできるか



パラメータの値ごとのMRR指標で
順位付け性能を評価

- ベースライン
→ Bug2Commit, FBL-BERT

- どのパラメータ設定でもベースラインを上回る
- $\lambda = 0.0$ (時間による減衰重みなし) よりも重みありが良い性能

RQ1 | FONTEはどの程度正確にBICを順位付けできるか

| 最適なパラメータ値

– $\alpha = 0$, $\tau = \max$, $\lambda = 0.1$

全ての順位設定でFONTEが優位

		All (# subjects = 130)					From Wen et al. [4] (# subjects = 67)					Manually Curated (# subjects = 63)						
Metric	MRR	Accuracy					MRR	Accuracy					MRR	Accuracy				
		@1	@2	@3	@5	@10		@1	@2	@3	@5	@10		@1	@2	@3	@5	@10
FONTE	0.528	47 (36%)	66 (51%)	85 (65%)	98 (75%)	110 (85%)	0.324	9	19	29	38	47	0.745	38	47	56	60	63
Other Techniques (on C_{BIC})																		
Bug2Commit	0.380	27	42	64	85	96	0.235	7	13	19	26	33	0.534	20	29	45	59	63
FBL-BERT	0.338	27	40	47	69	90	0.158	5	9	11	14	27	0.529	22	31	36	55	63
Random Baseline	0.218	3	19	41	65	75	0.065	0	2	4	6	12	0.381	3	17	37	59	63
Theoretical Lower Bound	0.145	3	8	19	41	71	0.039	0	1	2	4	8	0.258	3	7	17	37	63
Other Techniques (on C)																		
Bug2Commit	0.155	11	18	22	25	39	0.123	4	7	9	11	16	0.189	7	11	13	14	23
FBL-BERT	0.037	1	3	5	7	10	0.037	1	2	2	3	4	0.036	0	1	3	4	6
Random Baseline	0.002	0	0	0	0	0	0.002	0	0	0	0	0	0.002	0	0	0	0	0
Theoretical Lower Bound	0.001	0	0	0	0	0	0.001	0	0	0	0	0	0.001	0	0	0	0	0
Ablation Study for FONTE																		
Skip Stage 2	0.490	39	64	82	97	110	0.317	9	18	28	37	47	0.675	30	46	54	60	63
Use Equal Vote (No FL)	0.436	39	56	67	79	88	0.193	7	9	12	19	25	0.694	32	47	55	60	63
Max Aggr. (Eq. 9)	0.317	17	36	50	73	97	0.142	0	5	9	18	34	0.503	17	31	41	55	63

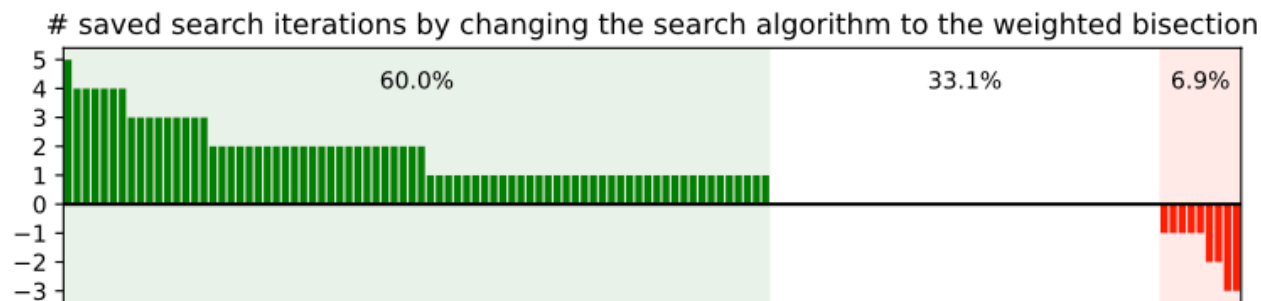
RQ2 | 重み付2分探索 VS 重みなし2分探索

| 重み付2分探索が効率的

- 平均67%の反復回数, 最大11回の反復を節約可能

| 反復回数の増減

- 統計的に, 重み付2分法の方が反復回数が少ない
 - 片側Wilcoxon 符号順位検定



まとめ

| 目的

- 正確, 効率的, デバッグ時に利用可能なBIC特定手法の提案

| FONTE

- テストカバレッジの使用, スタイル変更コミットの除外
 - 効率的な探索を実現
- バグレポートの不使用
 - デバッグ時に使用可能
- FL手法・時間による減衰重み
 - ベースラインを上回る, 正確なBICの特定

所感

| ランキングの正確性の意味がわからない

- BICは1つでは？
- データセットも不審度順に順位付けされている？

| FLスコアをこの式でコミットレベルにスケーリングできる意味がわからない

$$vote(e) = \frac{\alpha * susp(e) + (1 - \alpha) * 1}{rank_{\tau}(e)}$$

補足資料

| プログラムPの特性

- Pに対して行われたコミットの集合 : $C = \{c1, c2, \dots\}$
- Pのコード要素の集合 : $E = \{e1, e2, \dots\}$
- テストケースの集合 : $T = \{t1, t2, \dots\}$
 - 失敗したテストケースの集合 : $TF \subseteq T$
- テストtの実行中にeをカバー : $(t, e) \in \text{Cover}$
- コミットcがコード要素eの変更履歴に存在する : $(c, e) \in \text{Evolve}$